Zhaohui Wu
Chun Chen
Minyi Guo
Jiajun Bu (Eds.)

LNCS 3605

Embedded Software and Systems

First International Conference, ICESS 2004
Hangzhou, China, December 2004
Revised Selected Papers

Springer

Lecture Notes in Computer Science 3605

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Zhaohui Wu Chun Chen Minyi Guo
Jiajun Bu (Eds.)

Embedded Software and Systems

First International Conference, ICESS 2004
Hangzhou, China, December 9-10, 2004
Revised Selected Papers

Springer

Volume Editors

Zhaohui Wu
Chun Chen
Jiajun Bu
Zhejiang University, College of Computer Science
Hangzhou, 310027 P.R. China
E-mail: {wzh,chenc,bjj}@cs.zju.edu.cn

Minyi Guo
The University of Aizu Tsuruga
School of Computer Science and Engineering
Ikki-machi Aizu-Wakamatsu, Fukushima 965-8580, Japan
E-mail: minyi@u-aizu.ac.jp

Library of Congress Control Number: 2005932310

CR Subject Classification (1998): C.3, C.2, C.5.3, D.2, D.4, H.4

ISSN 0302-9743
ISBN-10 3-540-28128-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28128-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 11535409 06/3142 5 4 3 2 1 0

Preface

Welcome to the post proceedings of the First International Conference on Embedded Software and Systems (ICESS 2004), which was held in Hangzhou, P. R. China, 9–10 December 2004.

Embedded Software and Systems technology is of increasing importance for a wide range of industrial areas, such as aerospace, automotive, telecommunication, and manufacturing automation. Embedded technology is playing an increasingly dominant role in modern society. This is a natural outcome of amazingly fast developments in the embedded field.

The ICESS 2004 conference brought together researchers and developers from academia, industry, and government to advance the science, engineering, and technology in embedded software and systems development, and provided them with a forum to present and exchange their ideas, results, work in progress, and experience in all areas of embedded systems research and development.

The ICESS 2004 conference attracted much more interest than expected. The total number of paper submissions to the main conference and its three workshops, namely, Pervasive Computing, Automobile Electronics and Tele-communication, was almost 400, from nearly 20 countries and regions. All submissions were reviewed by at least three Program or Technical Committee members or external reviewers. It was extremely difficult to make the final decision on paper acceptance because there were so many excellent, foreseeing, and interesting submissions with brilliant ideas. As a result of balancing between accepting as many papers as possible and assuring the high quality of the conference, we finally decided to select 80 papers for the post-proceeding. We firmly believe that these papers not only present great ideas, inspiring results, and state-of-the-art technologies in recent research activities, but will also propel future developments in the Embedded Software and Systems research field.

The magnificent program for this conference was the result of the hard and excellent work of many people. We would like to express our sincere gratitude to all authors for their valuable contributions and to our Program/Technical Committee members and external reviewers for their great inputs and hard work. We are particularly grateful to our workshop chairs: Xiangqun Chen, Zhanglong Chen, Yue Gao, Xiaoge Wang, Xingshe Zhou and Mingyuan Zhu for their invaluable work in organizing wonderful workshops. We would also like to express our thanks to Jiajun Bu, Tianzhou Chen, Kougen Zheng, Minde Zhao, Hui Zhu, Shuying Tian, Fengxian Li and Cheng Jin for putting everything together to create this magnificent scientific event.

June 2005

Chun Chen, Xiangke Liao
Zhaohui Wu, Ranfun Chiu

Organization

ICESS 2004 was organized by Zhejiang University, Important Software Committee of the National 863 Program, the China Computer Federation, and the Hangzhou Association for Science and Technology.

Executive Committee

Honorary Chair	**Yunhe Pan**, Zhejiang University, China
General Chairs	**Xiangke Liao**, 863 Program Expert, China **Chun Chen**, Zhejiang University, China
Program Chairs	**Zhaohui Wu**, Zhejiang University, China **Ranfun Chiu**, HP lab, USA
Workshop Chairs	**Minyi Guo**, The University of Aizu, Japan **Kougen Zheng**, Zhejiang University, China
Public Relation Chair	**Tianzhou Chen**, Zhejiang University, China
Publication Chair	**Jiajun Bu**, Zhejiang University, China
Local Organizing Committee	**Minde Zhao (Chair)** **Shuying Tian, Hui Zhu, Fengxian Li, Cheng Jin, Wei Chen**

Sponsoring Corporations

Intel Corporation
China Putian Corporation
Hopen Software Eng. Co. Ltd.
ZTE Corporation
Huawei Technologies Co. Ltd.
CoreTek Systems Incorporated
China Mobile Software League

Program Committee

Makoto Amamiya	Kyushu University, Japan
Jiamei Cai	Zhejiang University of Industry, China
Tak-Wai Chan	National Central University, China
Xiangqun Chen	Peking University, China
Yaowu Chen	Zhejiang University, China
Zhanglong Chen	Fudan University, China
Walter Dosch	Medizinische Universität Lübeck, Germany
Nikil Dutt	University of California, Irvine, USA
Jesse Z. Fang	Intel Labs, USA
Yue Gao	Hopen Software Eng. Co. Ltd. China
Naiping Han	Chinasoft Network Technology Co., Ltd., China
R. Nigel Horspool	University of Victoria, Canada
Chris Hsiung	Hewlett-Packard Lab, USA
Margarida Jacome	University of Texas at Austin, USA
Moon Hae Kim	Konkuk University, Korea
Insup Lee	University of Pennsylvania, USA
Meng Lee	Hewlett-Packard Lab, USA
Xinming Li	ACET, China
Kwei-Jay Lin	University of California, Irvine, USA
Lei Luo	University of Electronic Science and Technology of China, China
Yi Pan	Georgia State University, USA
Xian-he Sun	Illinois Institute of Technology, USA
Walid Taha	Rice University, USA
Xiaoge Wang	Tsinghua University, China
Xing Zhang	Peking University, China
Xingshe Zhou	Northwestern Polytechnical University of China, China

Workshop Chairs

The International Workshop on Embedded Systems and Pervasive Computing

Xiangqun Chen	Peking University, China
Xingshe Zhou	Northwestern Polytechnical University, China

The International Workshop on Embedded Systems and Automobile Electronics

Zhanglong Chen	Fudan University, China
Mingyuan Zhu	CoreTek Systems Incorporated, China

The International Workshop on Embedded Systems in Telecommunication

Yue Gao	Hopen Software Eng. Co. Ltd., China
Xiaoge Wang	Tsinghua University, China

Technical Committee

Hamid R. Arabnia	University of Georgia, USA
Alessandro Bogliolo	University of Urbino, Italy
Luciano Bononi	University of Bologna, Italy
Jiajun Bu	Zhejiang University, China
Rajkumar Buyya	The University of Melbourne, Australia
Jiannong Cao	Hong Kong Polytechnic University, China
Tiziana Calamoneri	University of Rome "La Sapienza", Italy
Adriano Mauro Cansian	State University of Sao Paulo, Brazil
Naehyuck Chang	Seoul National University, Korea
Vipin Chaudhary	Wayne State University, USA
Shu-Ching Chen	Florida International University, USA
Shuoying Chen	Beijing Institute of Technology, China
Tianzhou Chen	Zhejiang University, China
Wenzhi Chen	Zhejiang University, China
Yu Chen	Tsinghua University, China
Zièd Choukair	ENST Bretagne, France
Hao-hua Chu	National Taiwan University, China
Chen Ding	University of Rochester, UK
PeiYu Fang	DYNA Technology, China
Feng Gao	Zhejiang University, China
Dan Grigoras	University College Cork, Ireland
Jianjun Hao	Beijing University of Posts and Telecommunications, China
Hangen He	Changsha Institute of Technology, China
Qianhua He	South China University of Technology, China
Yan Hu	China Electronics Standardization Institute, China
Liqun Huang	Huazhong University of Science and Technology, China
Zhiping Jia	Shandong University, China
Xiaohong Jiang	JAIST, Japan
Qun Jin	Waseda University, Japan
Mahmut Taylan Kandemir	Pennsylvania State University, USA
Ryan Kastner	University of California, Santa Barbara, USA
Dieter Kranzlmüller	University of Linz, Austria
Mohan Kumar	The University of Texas at Arlington, USA
Hsien-Hsin (Sean) Lee	Georgia Institute of Technology, USA
Trong-Yen Lee	National Taiwan University, China
Qing Li	City University of Hong Kong, China

Technical Committee (continued)

Jinlong Lin	Peking University, China
Youn-Long Steve Lin	National Tsing Hua University, China
Jilin Liu	Zhejiang University, China
Xiang Liu	GRAND Software, China
Xiang Liu	Peking University, China
Yan Liu	Putian-Smartcom, China
Zhaodu Liu	Beijing Institute of Technology, China
Zhen Liu	Nagasaki Institute of Applied Science, Japan
Bin Luo	Nanjing University, China
Lei Luo	CoreTek Systems Incorporated, China
Jingjian Lv	Beijing Open Lab (BOL) System Inc., China
HongBing Ma	Tsinghua University, China
Joberto Sérgio Barbosa Martins	University of Salvador, Brazil
Malena Mesarina	HP Labs, USA
Marius Minea	Universitatea Politehnica din Timişoara, Romania
Tatsuo Nakajima	Waseda University, Japan
Stephan Olariu	Old Dominion University, USA
Mohamed Ould-Khaoua	University of Glasgow, UK
Victor Prasanna	University of Southern California, USA
Huabiao Qin	South China University of Technology, China
Omer Rana	Cardiff University, UK
Edwin Sha	University of Texas at Dallas, USA
Lihong Shang	Beijing University of Aeronautics and Astronautics, China
Beibei Shao	Tsinghua University, China
Xiumin Shi	Beijing Institute of Technology, China
Timothy K. Shih	Tamkang University, China
Gurdip Singh	Kansas State University, USA
Zechang Sun	Tongji University, China
Zhenmin Tang	Nanjing University of Science and Technology, China
Pin Tao	Tsinghua University, China
Lorenzo Verdoscia	ICAR, CNR, Italy
Cho-li Wang	The University of Hong Kong, China
Dongsheng Wang	Tsinghua University, China
Farn Wang	National Taiwan University, China
Lei Wang	Beijing University of Aeronautics and Astronautics, China
Qing Wang	Institute of Software, Chinese Academy of Sciences, China
Guowei Wu	Dalian Institute of Technology, China

Technical Committee (continued)

Jie Wu	Florida Atlantic University, USA
Yong Xiang	Tsinghua University, China
Mingbo Xiao	Xiamen University, China
Cheng-Zhong Xu	Wayne State University, USA
Weikang Yang	Tsinghua University, China
Yanjun Yang	Peking University, China
Binyu Zang	Fudan University, China
Chengcui Zhang	University of Alabama at Birmingham, USA
Guobao Zhang	Southeast University, China
Jong Zhang	Beijing University of Aeronautics and Astronautics, China
Youtao Zhang	The University of Texas at Dallas, USA
Lin Zhong	Princeton University, USA
Huiyang Zhou	University of Central Florida, USA
Dakai Zhu	University of Pittsburg, USA

Table of Contents

Keynote Speeches and Invited Talks Abstracts (Partial)

Track 1 Distributed Embedded Computing

Track 2 Embedded Systems

Track 3 Embedded Hardware and Architecture

Track 4 Middleware for Embedded Computing

Track 5 Mobile Systems

Track 6 Transducer Network

Track 7 Embedded Operating System

Track 8 Power-Aware Computing

Track 9 Real-Time System

Track 10 Embedded System Verification and Testing

Track 11 Software Tools for Embedded Systems

Abstraction and the C++ Machine Model

Bjarne Stroustrup

Texas A&M University
(and AT&T Labs – Research)
http://www.research.att.com/~bs

Abstract. C++ was designed to be a systems programming language and has been used for embedded systems programming and other resource-constrained types of programming since the earliest days. This paper will briefly discuss how C++'s basic model of computation and data supports time and space performance, hardware access, and predictability. If that was all we wanted, we could write assembler or C, so I show how these basic features interact with abstraction mechanisms (such as classes, inheritance, and templates) to control system complexity and improve correctness while retaining the desired predictability and performance.

Ideals and Constraints

C++ [6] [10] is used in essentially every application areas, incl. scientific calculations, compilers, operating systems, device drivers, games, distributed systems infrastructure, animation, telecommunications, embedded systems applications (e.g. mars rover autonomous driving), aeronautics software, CAD/CAM systems, ordinary business applications, graphics, e-commerce sites, and large web applications (such as airline reservation). For a few examples of deployed applications, see http://www.research.att/~bs/applications.html.

How does C++ support such an enormous range of applications? The basic answer is: "by good use of hardware and effective abstraction". The aim of this paper is to very briefly describe C++'s basic model of the machine and how it's abstraction mechanisms map a user's high-level concepts into that model without loss of time of space efficiency. To put this in context, we must first examine the general ideals for programming that C++ is designed to support:

- Work at the highest feasible level of abstraction

Code that is expressed directly using the concepts of the application domain (such as band diagonal matrices, game avatar, and graphics transforms) is more easy to get correct, more comprehensible, and therefore more maintainable than code expressed using low-level concepts (such as bytes, pointers, data structures, and simple loops). The use of "feasible" refers to the fact that the expressiveness of the programming language used, the availability of tools and libraries, the quality of optimizers, the size of available memory, the performance of computers, real-time constraints, the background of programmers, and many other factors can limit our adherence to this ideal. There are still applications that are best written in assembler or very-low-level

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 1-13, 2005.

C++. This, however, is not the ideal. The challenge for tool builders is to make abstraction feasible (effective, affordable, manageable, etc.) for a larger domain of applications.

By "abstract", I do not mean "vague" or "imprecise". On the contrary, the ideal is one-to-one correspondence between application concepts and precisely defined entities in the source code:

- Represent concepts directly in code
- Represent independent concepts independently in code
- Represent relationships among concepts directly in code
- Combine concepts freely in code when (and only when) combination makes sense

Examples of "relationships among concepts" are hierarchies (as used in object-oriented programming) parameterized types and algorithms (as used in generic programming).

This paper is about applying these ideas to embedded systems programming, and especially to hard-real time and high-reliability embedded systems programming where very low-level programming techniques traditionally have been necessary.

What's special about embedded systems programming? Like so many answers about programming, this question is hard to answer because there is no generally accepted definition of "embedded systems programming". The field ranges from tiny controllers of individual gadgets (such as a car window opener), through stereo amplifiers, rice cookers, and digital cameras, to huge telephone switches, and whole airplane control systems. My comments are meant to address all but the tiniest systems: there can be no ISO C++ on a 4-bit micro-processor, but anything larger than that could potentially benefit from the ideals and techniques described here. The keys from a system design view are

- The system is not just a computer
  - It's a "gadget"/system containing one or more computers
- Correctness
  - "but the hardware misbehaved" is often no excuse
- Reliability requirements
  - Are typically more stringent than for an "ordinary office application"
- Resources constraints
  - Most embedded systems suffer memory and/or time constraints
- Real time constraints
  - Hard or soft deadlines
- No operator
  - Just users of "the gadget"
- Long service life
  - Often a program cannot be updates for the years of life of its gadget
- Some systems can't be taken down for maintenance
  - Either ever or for days at a time

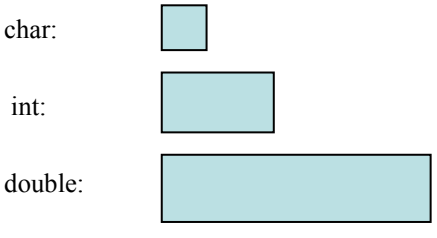
What does C++ have to offer in this domain that is not offered by assembler and C? In particular, what does the C++ abstraction mechanisms offer to complement the model of the machine that C++ shares with C? For a discussion of the relationship between C and C++, see [11].

Machine Model

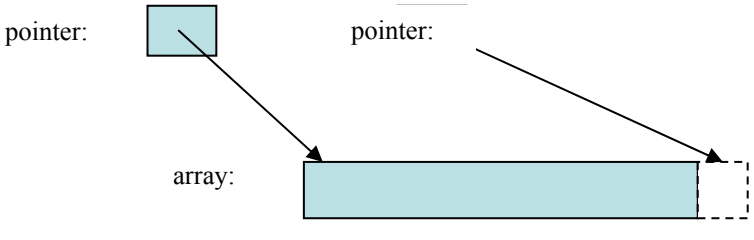
C++ maps directly onto hardware. Its basic types (such as **char**, **int**, and **double**) map directly into memory entities (such as bytes, words, and registers), most arithmetic and logical operations provided by processors are available for those types. Pointers, arrays, and references directly reflect the addressing hardware. There is no "abstract", "virtual" or mathematical model between the C++ programmer's expressions and the machine's facilities. This allows relatively simple and very good code generation. C++'s model, which with few exceptions is identical to C's, isn't detailed. For example, there is nothing in C++ that portably expresses the idea of a 2nd level cache, a memory-mapping unit, ROM, or a special purpose register. Such concepts are hard to abstract (express in a useful and portable manner), but there is work on standard library facilities to express even such difficult facilities (see the ROMability and hardware interface sections of [7]). Using C++, we can get really close to the hardware, if that's what we want.

Let me give examples of the simple map from C++ types to memory. The point here is not sophistication, but simplicity.

Basic arithmetic types are simply mapped into regions of memory of suitable size. A typical implementation would map a **cha**r to a byte, an **int** to a word, and a **double** to two words:



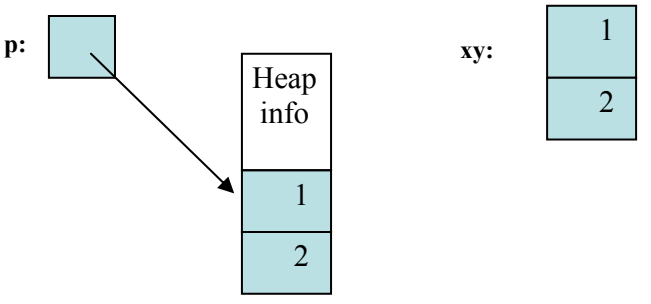
The exact map is chosen so as to be best for a given type of hardware. Access to sequences of objects is dealt with as arrays, typically accessed through pointers holding machine addresses. Often code manipulating sequences of objects deal with a pointer to the beginning of an array and a pointer to one-beyond-the-end of an array:



The flexibility of forming such addresses by the user and by the code generators can be important.

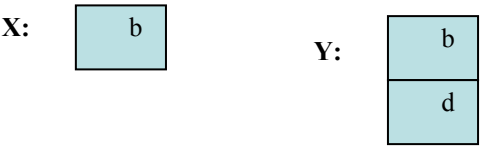
User-defined types are created by simple composition. Consider a simple type **Point**:

```
class Point { int x; int y; /* … */ };
Point xy(1,2);
Point* p = new Point(1,2);
```



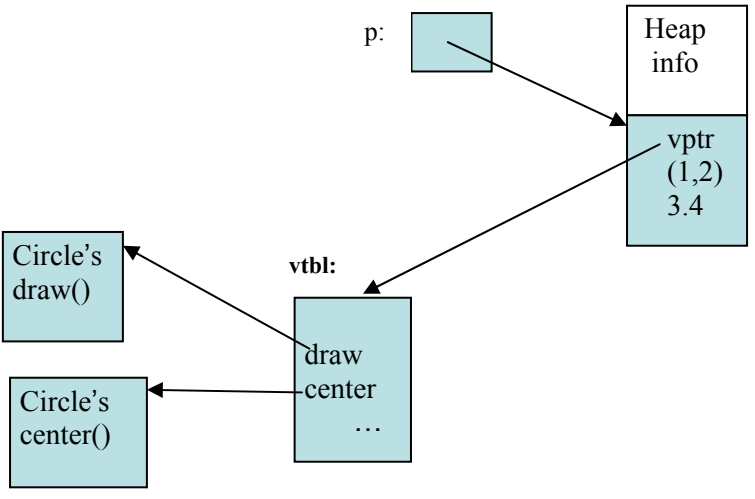
A **Point** is simply the concatenation of its data members, so the size of the **Point xy** is simply two times the size of an **int**. Only if we explicitly allocate a **Point** on the free store (the heap), as done for the **Point** pointed to by **p,** do we incur memory overhead (and allocation overhead). Similarly, basic inheritance simply involves the concatenation of members of the base and derived classes:

```
class X { int b; }
class Y : public X { int d; };
```



Only when we add virtual functions (C++'s variant of run-time dispatch supplying run-time polymorphism), do we need to add supporting data structures, and those are just tables of functions:

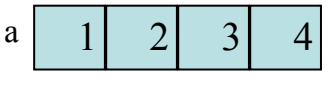
```
class Shape {
public:
        virtual void draw() = 0;
        virtual Point center() const = 0;
        // …
};

Class Circle : public Shape {
        Point c;
        double radius;
public:
        void draw() { /* draw the circle */ }
        Point center() const { return c; }
        // …
};

Shape* p = new Circle(Point(1,2),3.4);
```



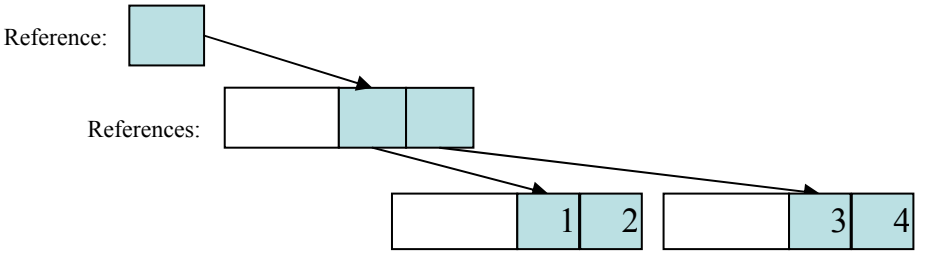
Naturally, this simple picture leaves out a lot, but when it comes to estimating time and space costs it's pretty accurate: What you see is what you get. For more details see [7]. In general, C++ implementations obey the zero-overhead principle: What you don't use, you don't pay for [8]. And further: What you do use, you couldn't hand code any better.

Please note that not every language provide such simple mappings to hardware and obeys these simple rules. Consider the C++ layout of an array of objects of a user-defined type:

```
class complex { double re, im; /* … */ };
complex a[ ] = { {1,2}, {3,4} };
```



The likely size is **4*sizeof(double)** which is likely to be 8 words. Compare this with a more typical layout from a "pure object-oriented language" where each user-defined object is allocated separately on the heap and accessed through a reference:



The likely size is **3*sizeof(reference)+3*sizeof(heap_overhead)+ 4*sizeof (double)**. Assuming a reference to be one word and the heap overhead to be two words, we get

a likely size of 17 words to compare to C++'s 8 words. This memory overhead comes with a run-time overhead from allocation and indirect access to elements. That indirect access to memory typically causes problems with cache utilization and limits ROMability.

Myths and Limitations

It is not uncommon to encounter an attitude that "if it's elegant, flexible, high-level, general, readable, etc., it must be slow and complicated". This attitude can be so ingrained that someone rejects essentially every C++ facility not offered by C without feeling the need for evidence. This is unfortunate because the low-level alternative involves more work at a lower level of abstraction, more errors, and more maintenance headaches. Bit, byte, pointer, and array fiddling should be the last resort rather than the first choice. C++ balances costs with benefits for "advanced features", such as classes, inheritance, templates, free store (heap), exceptions, and the standard library. If you need the functionality offered by these facilities, you can rarely (if ever) provide better hand-coded alternatives. The ISO C++ standard committee's technical report on performance [7] is provides data and arguments for that proposition.

Obviously, we should not use every feature of C++ for every problem. In particular, not every feature is suitable for hard real time because their performance is not 100% predictable (that is, we can't state in advance exactly how much an operation cost without knowing how it is used and the/or state of the program when it is used). The operations with this problem are:

- Free store (**new**/**delete**): The time needed for an allocation depends on the amount of available free memory and fragmentation can cause deterioration of performance over time. This implies that for many systems, free store cannot be used or can be used only at startup time (no deallocation implies no fragmentation). Alternatives are static allocation, stack allocation, and use of storage pools.
- RTTI (**dynamic_cast**/**typeid**): This is rarely needed in small embedded systems, so just don't use it for such systems. It is possible to implement **dynamic_cast** to be fast and predictable [3] but current implementations don't implement this refinement.
- Exceptions (**throw**/**catch**): The time needed to handle an exception depends on the distance (measured in function calls) from the throw-point to the catch-point and the number of objects needed to be destroyed on the way. Without suitable tools that's very hard to predict, and such tools are not available. Consequently, I can't recommend exceptions for hard real time; doing so is a research problem, which I expect to be solved within the decade. For now, we must use more conventional error-handling strategies when hard real time is needed, and restrict the use of exceptions to large embedded systems with soft real time requirements.

The rest of C++ (including classes, class hierarchies, and templates) can be used and has been used successfully for hard real time code. Naturally, this requires understanding of the facilities and their mapping to hardware, but that's no different

from other language constructs. Writing code for hard-real-time or high-reliability systems also requires caution and a good compiler (see http://www.research.att/~bs/compilers.html). It is worth noting that for many styles of usage, modern exception implementations are within 5% of the performance of non-exception code – and that non-exception code must be augmented with alternative exception-handling code (returning error codes, explicit tests, etc.). For systems where exceptions can be used, I consider them the preferred basis for an error-handling strategy [10].

Compilers used for embedded systems programming have switches to disable features where they are undesirable (e.g. in a hard-real time application). Anyway, their use is obvious from the source code.

Abstraction Mechanisms

The main abstraction mechanisms provided by C++ are classes, inheritance of classes, and templates. Here, I'll concentrate on templates because they are the key tool for modern statically type-safe high-performance code. Templates are a compile-time composition mechanism implying no runtime or space cost compared to equivalent hand-written code. Templates allow you to parameterize classes and functions with types and integers. If you like fancy words, they provide parametric polymorphism complementing the ad-hoc polymorphism offered by class hierarchies. Generally, systematic use of templates is called "generic programming" which complements the "object-oriented programming" that you get from systematic use of class hierarchies. Both programming paradigms rely on classes.

I will first present some simple "textbook examples" to illustrate the general techniques and tradeoffs. After that, I'll show some real code from a large marine diesel engine using those same facilities to provide reliability, safety, and performance.

Here is a slightly simplified version of the C++ standard library complex type. This is a template class parameterized by the scalar type used:

```
template<class Scalar>
class complex {
    Scalar re, im;
public;
    complex() { }
    complex(Scalar x) : re(x) { }
    complex(Scalar x, Scalar y) : re(x), im(y) { }

    complex& operator+=(complex z) { re+=z.re; im+=z.im; return
*this; }
    complex& operator+=(Scalar x) { re+=x; return *this; }

     // …
};
```

This is a perfectly ordinary class definition, providing data members (defining the layout of objects of the type) and function members (defining valid operations). The

template<class Scalar> says that **complex** takes a type argument (which it uses as its scalar type). Given that definition – and nothing else – we can write

```
complex<double> z(1,2);        // z.re=1; z.im=2;
complex<float> z2 = 3;         // z2.re=3;

z += z2;                       // z.re=z.re+z2.re; z.im=z.im+z2.im;
```

The comments indicate the code generated. The point is that there is no overhead. The operations performed are at the machine level exactly those required by the semantics. A **complex<double>** is allocated as two **double**s (and no more) whereas a **complex<float>** is allocated as two **float**s. A **complex<int>** would make a rather good **Point** type. No code or space is generated for the template class **complex** itself and since we didn't use the **+=** operation taking a scalar argument, no code is generated for that either. Given a decent optimizer, no code is laid down for the used **+=** operation either. Instead, all the operations are inlined to give the code indicated in the comments.

There are two versions of += to ensure optimal performance without heroic efforts from the optimizer. For example, consider:

```
z+=2;                  // z.re+=2
z+=(2,0);      // z.re+=2; z.im+=0;
```

A good optimizer will eliminate the redundant **z.im+=0** in the second statement. However, by providing a separate implementation for incrementing only the real part, we don't have to rely on the optimizer to be that clever. In this way, overloading can be a tool for performance.

We can use the += operation to define a conventional binary +:

```
template<class S>
complex<S> operator+(complex<S> x, complex<S> y)
{
        complex<S> r = x;      // r.re=x.re; r.im=y.im;
        r+=y;                  // r.re+=y.re; r.im+=y.im;
}

// define complex variables x and y
complex<double> z = x+y;       // z.re=x.re+y.re; z.im=x.im+y.im;
```

Again the comments indicate the optimal code that existing optimizers generate for this. Basically, the templates map to the implementation model for classes described above to give good use of memory and inlining of simple function calls ensures good use of that memory. By "good" I mean "optimal given a good optimizer" and optimizers that good are not uncommon. The example above might make a simple first test of your compiler and optimizer if you want to see whether it is suitable for an application.

The performance of this code depends on inlining of function calls. It has correctly been observed that inlining can lead to code bloat when a large function is inlined many times (either for many different calls or for s few calls but with different template arguments). However, that argument does not apply to small functions (such as, the += and + defined for **complex**) where the actual operation is smaller and faster than the function preamble and value return. In such cases, inlining provides improvements in both time and space compared with ordinary functions and ordinary function calls. In fact, a popular use of class objects and inline function is to implement parameterization where the parameter can be a single machine instruction, such as < [9].

Inlining a large function is usually a very bad idea. Doing so typically indicates carelessness on behalf of the programmer or a poorly tuned optimizer.

In sharp contrast to the claim that templates cause code bloat, it so happens that templates can be used to save code space. A C++ compiler is not allowed to generate code for an unused template function. This implies that if a program uses only 3 of a template class' 7 member functions, only those three functions will occupy space in memory. The equivalent optimization for non-template classes is not common (the standard doesn't require it) and extremely hard to achieve for virtual functions.

The perfect inlining of small member functions and the guarantee that no code is generated for unused functions is the reason that function objects have become the preferred way of parameterizing algorithms. A function object is an object of a class with the application operator () defined to perform a required action. For example

```
template<class T> struct less {
        bool operator()(const T& a, const T& b) const { return a<b; }
};
```

This function object, **less**, is used by most standard library facilities that need to perform a comparison. The result can be factors of improvement in run time compared to parameterization with a function pointers for algorithms such as **sort()** [Stroustrup, 1999].

Most uses of templates are described as "generic programming" or "template meta-programming". Both are based on overloading where we let the compiler pick the right implementation based on types (and integer values). The simplest and most familiar example is the compiler choosing the right implementation of + when we add **int**, **double**, **complex**, etc. values. The compiler can pick the right function (or basic operation) based on argument types. Similarly, the compiler will pick the right type for an object based on template arguments.

The selection of types and operations is done at compile time and can lead to major improvements. For example, in an embedded application the indirection through pointers to manipulate device drivers turned out to be the bottleneck. The solution was to replace hand-optimized low-level C with templates parameterized on the device register addresses and object types; a 40% improvement in performance was achieved that way. The resulting code was also much shorter and easier to maintain [5]. Section 5 of [7] contains code illustrating such techniques; the examples there relate to a standard interface to special-purpose registers.

It's amazing what you can do using these techniques. One place to look for techniques and examples is the STL (the C++ standard library's framework for

containers and algorithms) [10]. Since the STL relies on free store it may not be applicable to your particular embedded application, but the techniques are general. For more advanced/extreme uses labeled "template metaprogramming", see [1] and for lots of examples see the Boost collection of libraries [2].

For generality, it is important that templates can have integer arguments. In particular, you can do arbitrary computations at compile time; compile-time constant folding is just the simplest example.

Code Examples

Consider briefly a problem faced by the designers of control and monitoring software for large (100,000Hp+) marine diesel engines at MAN B&W Diesel A/S. These engines simply can't be allowed to fail (or a huge ship is adrift), the engine computers must potentially work for years without maintenance, and programs must be portable to new generations of computers (since computer generations are shorted than engine generations) [4].

How can we compute accurately and safely? Using numbers of different accuracies? And detect errors such as dived by zero and overflow? Fast enough for hard-real time? (on rugged hardware based on 25MHz Motorola 68332 processors used for electronic fuel injection). The solution chosen and now running on huge ships on the high seas involves:

- Make a template class for fixed-point arithmetic
  - Fixed point is completely portable
  - Fixed point is most efficient on the relevant processors
- Use template specializations where needed

As expected and required, this solution has zero overhead in time and space.

Consider first an example of a function that performs a critical computation. I have done nothing to this code except adjusting the indentation. I am told that it is easy to read if you understand about the engine. Having seen far worse looking code for far simpler problems, I have no trouble believing that:

```
StatusType<FixPoint16> EngineClass::InternalLoadEstimation(
                                const      StatusType<FixPoint16>&
UnsigRelSpeed,
                                const      StatusType<FixPoint16>&
FuelIndex)
  {
      StatusType<FixPoint16> sl =UnsigRelSpeed*FuelIndex;

      StatusType<FixPoint16> IntLoad =
        sl*(PointSevenFive+sl*(PointFiveFour-PointTwoSeven*sl))
        -
PointZeroTwo*UnsigRelSpeed*UnsigRelSpeed*UnsigRelSpeed;

      IntLoad=IntLoad*NoFuelCylCorrFactor.Get();
```

```
        if (IntLoad.GetValue()<FixPoint16ZeroValue)
                IntLoad=sFIXPOINT16_0;

        return IntLoad;
    }
```

The 16-bit fixed point type is just an ordinary class:

```
    stuct FixPoint16 {
        FixPoint16();
        FixPoint16(double aVal);

        bool    operator==(const FixPoint16& a) const { return
val==a.val; }
        bool  operator!=(const FixPoint16&) const;
        bool  operator>(const FixPoint16&) const;
        bool  operator<(const FixPoint16&) const;
        bool  operator>=(const FixPoint16&) const;
        bool  operator<=(const FixPoint16&) const;

        short    GetShort() const;
        float    GetFloat() const;
        double  GetDouble() const;
    private:
        long      val;      // e.g. 16.16
    };
```

The real computation (of engine status) takes place on status types (parameterized by arithmetic types, such as **FixPoint16**):

```
    template <class T>
    struct StatusType {
      StatusType();
      StatusType(const StatusType&);
      StatusType(const T aVal,const unsigned long aStat);

      // Member Compound-assignment operator functions:
      StatusType& operator+=(const StatusType&);

      // Miscellaneous:
      const T& GetValue() const;

      // Access functions for status bits:
      bool isOk() const;
      bool IsValid() const;
    private:
      T value;
```

```
    unsigned long fpStatus;  // Bit codes defined by type tagFixPoint16Status
};
```

This template class is designed and implemented using the techniques we saw for **complex**. For its time and space performance, it relies on the same techniques and optimizations. This implies that the techniques (and the tools that supports them) are effective in real-world embedded systems contexts.

The low-level details of the engine and the processor are encoded in constants and encapsulated in functions relying on such constants:

```
template<class T>
inline bool StatusType<T>::IsValid() const
{
   return (bool)((fpStatus & 0x0000FFFF) ==VS_VALID);
}


template <>
StatusType<long>&
      StatusType<long>::operator+=(const StatusType<long>& rhs)
{
    long sum = value + rhs.value;

    if ((value ^ sum) & (rhs.value ^ sum) & LONG_MSB)  { // overflow
        AppendToStatus(VS_OVERFLOW);
        value = (sum & LONG_MSB ? LONG_MAX : LONG_MIN);
    }
    else {
        value = sum;
    }

    AppendToStatus(rhs.GetStatus());

    return (*this);
}
```

The designers of this software emphasize (my translation from Danish):

- C++ is not just used as ”A better C”
  - Our results far exceeded our outside consultants experience with comparable C-based projects.
- Heavy use of object-oriented techniques
  - Including class hierarchies and virtual functions
- Heavy use of generic programming and templates
  - Essential to avoid code duplication

  - Essential to achieve optimal performance
  - Object-oriented and generic programming used in combination
- A good tool chain is essential

The code does not use exceptions (since it is a hard-real-time program) and free store allocation is only used during startup where memory exhaustion and fragmentation cannot occur.

Acknowledgements

Special thanks to Mogens Hansen and Martin O'Riorden for making their examples available to me and educating me in some newer techniques used in performance-critical and safety-critical embedded systems programming. Also thanks to the members of the ISO C++ standard committee's Performance working group who collected the information for [7].

References

1. David Abrahams and Aleksey Gurtovoy: "C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond". Addison Wesley. 2005. ISBN 0-321-22725-5.
2. www.boost.org.
3. Michael Gibbs and Bjarne Stroustrup: "Fast Dynamic Casting". Software-Practice&Experience. Wiley. To appear 2005.
4. Mogens Hansen: "C++ I embedded systemer". Elektronik 04. Odense Congress Center. September 2004. And personal communication.
5. Martin J. O'Riordan: "C++ For Embedded Systems". And personal communications.
6. "The C++ Standard" (ISO/IEC 14882:2002). Wiley 2003. ISBN 0 470 84674-7.
7. "Technical Report on C++ Performance". ISO.IEC PDTR 18015. (http://www.research.att.com/ ~bs/performanceTR.pdf).
8. Bjarne Stroustrup: "The Design and Evolution of C++". Addison Wesley, 1994. ISBN 0-201-54330-3.
9. Bjarne Stroustrup: "Learning standard C++ as a new language". C/C++ Users Journal. May 1999
10. Bjarne Stroustrup: "The C++ Programming Language". Addison Wesley. 2000. ISBN 0-201-70073-5.
11. B. Stroustrup: "C and C++: Siblings", "C and C++: A Case for Compatibility", "C and C++: Case Studies in Compatibility". The C/C++

Keynote Speech: Industrializing Software Development

Alexander Stepanov

Adobe Systems
USA

The objective of the talk is to discuss economic, organizational, and technological aspects of software industrialization. While it is impossible to predict exactly when the industrial revolution in software will occur, it is clear that when it happens it will cause a dramatic redistribution of wealth and a decline of the software monopolies.

There is the economic reason why software components as an industry (predicted in the late sixties by Doug McIlroy) never materialized: it is the emergence of the software industry, whose very existence is based on unspecified, irregular and extremely complex interfaces.

Organizationally, there is no division of labor, a very low level of professionalism, and a reward system that is based on number of features, rather than on the level of reliability, correctness, and security.

Finally, technologically we still have to learn to produce comprehensive, well-organized catalogs of highly generic, reliable components with precise time and space performance characteristics.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 14-14, 2005.

Testing Methodologies for Embedded Systems and Systems-on-Chip

Laurence T. Yang[1] and Jon Muzio[2]

[1] Department of Computer Science, St. Francis Xavier University
P.O. Box 5000, Antigonish, B2G 2W5, NS, Canada
[2] Department of Computer Science, University of Victoria
Victoria BC, V8W 3P6 Canada

Abstract. Testing of a fabricated chip is a process that applies a sequence of inputs to the chip and analyzes the chip's output sequence to ascertain whether it functions correctly. As the chip density grows to beyond millions of gates, Embedded systems and systems-on-chip testing becomes a formidable task. Vast amounts of time and money have been invested by the industry just to ensure the high testability of products. On the other hand, as design complexity drastically increases, current gate-level design and test methodology alone can no longer satisfy stringent time-to-market requirements. The High-Level Test Synthesis (HLTS) system, which this paper mainly focuses on, is to develop new systematic techniques to integrate testability consideration, specially the Built-In Self-Test (BIST) methodology, into the synthesis process. It makes possible for an automatic synthesis tool to predict testability of the synthesized embedded systems or chips accurately in the early stage. It also optimizes the designs in terms of test cost as well as performance and hardware area cost.

1 Introduction

Driven by the rapid growth of the Internet, communication technologies, pervasive computing, automobiles, airplanes, wireless and portable consumer electronics, Embedded Systems and Systems-on-Chip (SoC) have moved from a craft to an emerging and very promising discipline in today's electronic industry.

Testing of a fabricated very large scale integrated embedded systems and system-on-chip is a process that applies a sequence of inputs to the circuit and analyzes the circuit's output sequence to ascertain weather it functions correctly. As the chip density grows to beyond millions of gates, testing becomes a formidable task. Vast amounts of time and money have been invested by the semiconductor industry just to ensure the high testability of products. A number of semiconductor companies estimate that about 7% to 10% of the total cost is spent in single device testing [17]. This figure can rise to as high as 20% to 30% if the cost of in-circuit testing and broad-level testing is added. However, the most important cost can be the loss in time-to-market due to hard-to-detect faults. Recent studies show that a six-month delay in time-to-market can cut profits

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 15–24, 2005.

by 34% [17]. Thus, testing can pose serious problems in embedded system and systems-on-chip designs.

Part of reason testing cost so much is the traditional separation of design and testing. Testing is often viewed inaccurately as a process that should start only after the design is complete. Due to this separation, the designer usually has little appreciation of testing requirements, whereas the test engineer has little input into the design process. In order to effectively reduce testing cost, methods which take into account testability of the final product are needed and are usually called *Test Synthesis*. This approach is motivated by the high complexity of current design and related testing costs. The design test related activities, such as test generation and test application, usually have a relatively big share of the total design and test cost. In some cases, this can reach to as high as 50% of the total cost. Thus the main idea of *Test Synthesis* is to improve testability of the design during early stages which is expected to reduce the later design testing costs.

On the other hand, as design complexity drastically increases, current gate-level synthesis methodology alone can no longer satisfy stringent time-to-market requirement. *High-level Synthesis* [2,5] which takes a behavioral specification of a digital system and a set of design constraints as input and generates a Register-Transfer Level (RTL) hardware implementation is hence considered as a promising technology to boost design quality and shorten the development cycle.

The main objective of the **High-level Test Synthesis** this paper focus on is to develop new systematic techniques to integrate testability consideration into synthesis process and make it possible for an automatic synthesis tool to predict testability of the synthesized circuits accurately in the early stage and optimize the designs in terms of test cost as well as performance and area cost.

2 Recent Research Summary

Due to the increasing gate-to-pin ratios which limit the feasibility of testing digital circuits externally, this paper mainly describes some recent research progress on our work of a built-in self-test synthesis system. Its framework is depicted in Figure 1.

2.1 Design Representation

First of all, our system takes a VHDL behavioral specification of a digital system and a set of design constraints as input and generates a Register-Transfer Level (RTL) hardware implementation which consists of a data path and a controller. The kernel of the system is an intermediate design representation, called Extended Timed Petri Net (ETPN), which can be used both for testability analysis and high-level synthesis [14]. In ETPN, the structural properties of the data path and controller are explicitly captured in order to facilitate accurate analysis of the intermediate design in term of performance, area and testability.

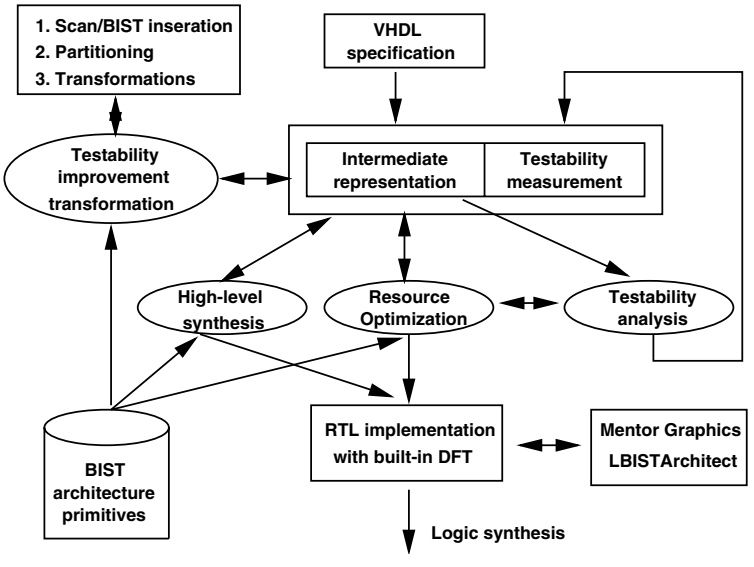


Fig. 1. The built-in self-test synthesis system

2.2 Data Path Testability Analysis

Based on the design representation, we have developed register transfer level data path testability metrics to evaluate various BIST configurations and make improvement decision [23,25]. The early decision about testability improvement gives the possibility that designs can be optimized in later synthesis processes. The testability analysis carried out at high-level abstraction will also reduce the computational complexity, since the complexity of a design at this level is significantly lower.

The objective of testability metrics is to analyze and quantify BIST testability for a given register transfer level design. Basically, our BIST testability metrics quantify two important testability aspects, namely controllability and observability. In our approach, we mainly follow the test scheme, namely minimal behavioral BIST originally proposed in [10]. Both of controllability and observability are further divided into two factors: combinational factor and sequential factor. The combinational factor is measured in terms of the quality of pseudo-random values as they propagate through embedded modules and registers, and the sequential factor is used for the estimated number of steps or clock cycle to control under test. Similarly the combinational observability is measured in terms of sensitivity of embedded modules and registers to erroneous value propagation, i.e. in terms of how difficult it is to propagate an erroneous value through to an observable output, and the sequential factor is used for the estimated number of steps or clock cycle to observe under test. As a result, our testability metric consists of, therefore, four measures:*combinational controllabil-*

ity (CC), *sequential controllability* (SC), *combinational observability* (CO) and *sequential observability* (SO) [31] based on Markov chain model [3], and provides a means of measuring the effect of test improvement with regards to BIST test quality.

2.3 State Reachability Analysis

Besides the data path testability metrics, we also have developed state reachability metrics which are used to characterize the testability of the given controller in term of a ETPN [23,25]. It is defined by the difficulty of reaching a state from an initial state. This measurement is associated with each state in the control part. The state reachability consists of two measurements, namely combinational state reachability (CSR) and sequential state reachability (SSR) [7]. The combinational state reachability measures the probability to reach the current state from an initial state, and the sequential state reachability measures the number of cycles (transitions) needed to reach the current state from the initial state.

2.4 Incremental Testability and Reachability

Due to the large computational complexity of testability and state reachability analysis and the need to perform such analysis after each synthesis steps, we have applied a similar systematic technique used for ATPG technique for the present BIST technique to approximate accurately the repeated testability and state reachability calculation and evaluation [23,25].

First of all, the global testability of a data path is based on a cost function in [9] and is used to estimate the global testability of an entire design. Based on the above global testability measurement, we propose a new and efficient estimation method [23,25] which is based partially on explicit re-calculation and partially on gradient techniques for incremental testability and state reachability to update the test property.

2.5 BIST Partitioning

Based on the above testability measurements, we develop a new improvement method with BIST technique at register transfer level(RTL). RTL circuits consist of interconnections of registers, functional units (ALUs), multiplexors and buses. Both conventional BIST [1] and circular BIST [15,16] are well-suited for automatic circuit improvement at the register transfer level. Traditionally, each ALU in a circuit is made directly testable by placing test registers to generate test patterns at the ALU's inputs, and the test registers to compact the responses at the ALU's output. However, it may not be necessary to add this many test registers [4]. For example, suppose that the input registers to the ALU are not directly controllable, but they still can generate patterns that are random enough to efficiently test the ALU; in this case, there is no need to replace the normal system registers with more expensive, slower test registers. Thus, an efficient

partitioning technique, which decide either which registers should be configured as test registers (conventional BIST) or which registers should be linked in the circular scan path (circular BIST), is necessary.

Partitioning for a design can lead to the simplifications of many design procedures such as synthesis and test. Partitioning for testability will lead to the simplification of test efforts and the ability to apply different test strategies to different partitions. The proposed partitioning technique in the paper [23,25] transforms some hard-to-test registers and/or lines to boundary components. These components act as normal registers and/or lines in the normal mode and serve as partitioning boundaries in test mode or test registers. Therefore, a design is partitioned into several sub-circuits and each of them can be tested and controlled based on BIST test schemes. It is, therefore, possible to apply different test strategies, such as scan for deterministic and BIST for random test to different partitions.

The circuit partitioning problem can, in general, be formulated as a graph partitioning problem. Given a graph with nodes and arcs, the objective is to partition the nodes into several subsets, such that the total costs of the arcs between nodes in different partitions is minimized. Optimal partitioning is known to be NP-complete [6]. In our research work, we present an efficient and economic BIST partitioning approach [23,25]. It is based on a BIST testability analysis algorithm with an incremental testability analysis approach for data path and a state reachability analysis algorithm with its incremental analysis approach for control path at register-transfer level. Initially we use the testability algorithm for data path and state reachability algorithm for control part to find partitioning boundaries. Then the partitioning procedure is performed quantitatively by a clustering algorithm which clusters directly interconnected components excluding boundary components based on the global testability of data path and global state reachability analysis of control part. After each selection step, we use the proposed new and efficient estimation method which is based partially on explicit re-calculation and partially on gradient techniques for incremental testability and state reachability to update the test property. This process will be iterated until the design is partitioned into several disjoint sub-circuits and each of them can be tested independently. Therefore, the design is fully self-testable.

2.6 Resource Optimization

Applying BIST techniques for resource optimization before going to RTL implementation or performing high-level synthesis involves modification of the hardware on the chip so that the chip has the capability to test itself. Table 1 in [20] shows different types of test registers that can be used. *Concurrent built-in logic observation (CBILBO) and built-in logic observation (BILBO)* registers can both generate test pattern and compress test response, and ensure high fault-coverage. BILBO registers need more test sessions while CBILBO registers require more hardware area. Note that a test register usually has larger hardware area than a normal register (see Table 1 in [20] where ω is the area scaling factor

over normal register). For example, CBILBOs have an area approximation twice that of the normal registers. One of the main considerations for BIST resource optimization is, therefore, the extra area for the test circuitry. Here we would like to describe an optimal modification approach [21] based on Integer Linear Programming formulation [11] to find BIST embeddings in the data path prepared for the synthesis algorithm or before going to RTL implementation such that the cost of modification is minimum.

2.7 Data Path Allocation

It has been shown that MISR registers can also be used to generate pseudo-random test patterns [8,18]. This results in both reduction of testing times and reduction of extra registers which reduces hardware area. However, since the actual time required for a MISR register to obtain exhaustive pattern coverage is exponential with respect to the number of bits in the register, as shown in [20], the test quality might be reduced. How to reduce this area overhead without sacrificing the test quality is one of the major concern of our research work.

Considering testability issues at high-level synthesis can lead to a more efficient exploration of the design space, thus resulting in a digital circuit that requires minimal BIST area overhead and has high test concurrency while guaranteeing the test quality.

The fact that the contents of signature registers (MISR) can be used as test patterns leads to the following advantages. First, the algorithm produces designs with high test concurrency which reduces the overall testing time due to increased testing parallelism. Moreover, the number of extra registers for implementing BIST can be reduced. However, since the actual time required for a MISR register to obtain exhaustive pattern coverage is exponential with respect to the number of bits in the register, we consider such template as *incompletely embedded* module. We describe a high-level data path allocation algorithm in [20] which generates highly testable data path designs while maximizing the sharing of modules and test registers. Module allocation is guided by a testability balance principle where *incompletely embedded* modules can be mapped into the same function module that is *completely embedded*. In this way, the *incompletely embedded* module after allocation will be fully testable. The register allocation is mainly based on the sharing degrees of registers which reflects the number of modules for which the register can be configured as RTPG and the number of modules for which it can be configured as a MISR. Using this measure the register allocation is guided by choosing mergers that result in large increases in the sharing degrees of registers over those resulting in smaller increases. This would result in registers with high sharing degrees, thereby requiring a smaller number of BIST registers globally in the design.

However, the approach still has some drawbacks, for example, if an *incompletely embedded* module can not find a match to be merged with a *completely embedded* module during the iterative allocation algorithm. It probably will not become fully testable. If there are several such modules un-mapped in the design, the resulting testing quality will be not satisfactory. This motivates us to make

use of two types of redundant transformations introduced in [11,12,13], which add redundancy that improves test resources to be shared in the data path without affecting the scheduling step (latency) and functional resource requirement of the behavior, to improve our data path allocation algorithm and to make all *incompletely embedded* modules become fully testable [27].

2.8 Integrated Synthesis Algorithm

After our system takes a VHDL behavioral specification of a digital system and a set of design constraints as input, the design representation is always unscheduled. Therefore, we need to consider not only operation scheduling but also data path allocation.

In our research work, we describe a high-level test synthesis algorithm for operation scheduling and data path allocation [24]. It generates highly testable data path designs while maximizing the sharing of test registers, which means that only a small number of registers is modified for BIST. The algorithm produces also designs with high test concurrency, thereby decreasing test time. The algorithm is motivated by that if the contents of signature registers can be used as test patterns, the overall testing time can be reduced due to increased testing parallelism, moreover, the number of extra registers for implementing BIST can be reduced. In our approach, module allocation is guided by a testability balance principle where *incompletely embedded* modules can be mapped into the same function module that is *completely embedded*. In this way, the *incompletely embedded* module after allocation will be fully testable. The register allocation is guided by an incremental sharing measurement which chooses merges that result in large increases in the sharing degrees of registers. When two modules are merged, the operations executed on these modules must be scheduled in different control steps so that they can share the same component. Similar for registers, the variables stored in these registers must be disjoint. We will present the rescheduling transformation which is performed by a merge-sort algorithm. These transformations change locally the execution orders of some operations in the current schedule in order to improve the testability and satisfy the scheduling constraints imposed by the merger. Contrary to other works in which the scheduling and allocation tasks are performed independently, our approach integrates scheduling and allocation by performing them simultaneously so that the effects of scheduling and allocation on testability are exploited more effectively.

In [22], we also introduce some concepts and techniques to improve our previous work [24] during the operation scheduling part, specially to determine the execution order of different operations when rescheduling transformations are performed.

However, the above described integrated approach still has some drawbacks, for example, during the module allocation, if an *incompletely embedded* module can not find a match to be merged with a *completely embedded* module during the iterative allocation algorithm. It probably will not become fully testable. If there are several such modules un-mapped in the design, the resulting testing quality will be not satisfactory. Similarly if a pair of operation in the same scheduling

step to be merged based on the allocation balance principle is decided, we have to introduce dummy places which have negative impacts or increase the control steps leading to longer execution time or slow performance. This motivates us to make use of two types of redundant transformations introduced in [11,13], which add redundancy that improves test resources to be shared in the data path without affecting the scheduling step (latency) and functional resource requirement of the behavior, to improve our data path allocation algorithm and to make all *incompletely embedded* modules become fully testable [28]. Also this can avoid the increase of scheduling steps because one of the operations can be merged with the introduced redundant operations at different scheduling steps. In [28] we have demonstrated the advantage of the approach by introducing the redundant transformations for operation scheduling and data path allocation.

2.9 Testability Metrics-Based Synthesis

In [19], we also present a different BIST synthesis methodology, namely a BIST testability metrics-based algorithm for operation scheduling and data path allocation. It is based on the BIST data path testability analysis algorithm at register-transfer level described in previous subsections. In the approach, module and register allocation are guided by a testability balance technique. In our approach, the selection of nodes to be merged is based on the testability measures generated by the testability analysis algorithm. The main goal is to generate a data path with good controllability and observability for all the nodes and with as few loops as possible. The basic idea is to fold nodes with good controllability and bad observability to nodes with good observability and bad controllability. Note that the controllability of a node is defined as the best controllability of any of its input lines. While the observability of a node is the best observability of any of its output lines. In this way, the new node will inherit the good controllability from one of the old nodes and the good observability from the other. The synthesis algorithm introduces scheduling constraints imposed by data path allocation and performs scheduling and allocation simultaneously in an iterative fashion so that their effects on testability are exploited more effectively.

With the help of an incremental BIST testability analysis and a state reachability analysis with its incremental approach for control path at register-transfer level, we mainly make use of some concepts and techniques to improve the above work [19] not only during the data path synthesis part, but also during the operation scheduling part [26].

Similarly redundant transformations have been introduced [29]. We add redundancy that improves test resources to be shared in the data path without affecting the scheduling step (latency) and functional resource requirement of the behavior, to improve our data path allocation algorithm and to make all modules become fully testable. Also this can avoid the increase of scheduling steps because one of the operations can be merged with the introduced redundant operations at different scheduling steps [30].

References

1. V. D. Agrawal, C. R. Kime, and K. K. Saluja. A tutotial on Build-in Self-test, part 1: principles. *IEEE Design and Test of Computers*, March 1993.
2. R. Camposano and W. H. Wolf. *High-Level VLSI Synthesis*. Kluwer Academic Publishers, 1991.
3. J. Carletta and C. A. Papachristou. Testability analysis and insertion for RTL circuits based on pseudorandom BIST. In *Proceedings of International Conference on Computer Design*, 1995.
4. S. Chiu and C. A. Papachristou. A design for testability scheme with applications to data path synthesis. In *Proceedings of Design Automation Conference*, pages 271–277, June 1991.
5. D. D. Gajski, N. D. Dutt, A. C-H. Wu, and Steve Y-L. Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., San Francisco, 1979.
7. X. Gu, E. Larsson, K. Kuchcinski, and Z. Peng. A controller testability and enhancement technique. In *Proceedings of European Design and Test Conference*, pages 153–157, Paris, France, March 1997.
8. K. Kim, D. S. Ha, and J. G. Tront. On using signature registers as pseudorandom pattern generators in built-in self-testing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(8):919–928, August 1988.
9. R. Lisanke, F. Braglez, A. J. Degues, and D. Gregory. Testability-driven random test pattern generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6:1082–1087, 1987.
10. C. A. Papachristou and J. Carletta. Test synthesis in the behavioral domain. In *Proceedings of International Test Conference*, October 1995.
11. I. Parulkar. *Optimization of BIST resource during high-level synthesis*. PhD thesis, University of South California, May 1998.
12. I. Parulkar, S. Gupta, and M. Breuer. Introducing redundant computations in a behavior for reducing BIST resources. In *Proceedings of the 35th ACM/IEEE Design Automation Conference (DAC-98)*, pages 548–553, San Francisco, USA, June 15-18, 1998.
13. I. Parulkar, S. Gupta, and M. Breuer. Introducing redundant computations in RTL data paths for reducing BIST resources. *ACM Transactions on Design Automation of Electronic Systems*, 6(3):423–445, 2001.
14. Z. Peng and K. Kuchcinski. Automated transformation of algorithms into register-transfer level implementations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 150–166, 1994.
15. S. Pilarski, A. Krasniewski, and T. Kameda. Estimating testing effectiveness of the circular self-test path technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(10):1301–1316, October 1992.
16. M. M. Pradhan, E. J. Brrien, S. L. Lam, and J. Beausang. Circular BIST with partial scan. In *Proceedings of International Test Conference*, pages 719–729, October 1988.
17. L. Rosqvist. *Application Specification Integrated Circuit (ASIC) Technology*, chapter 8, Test and testability of ASICs. Academic Press, San Diego, California, 1991.
18. L. T. Wang and E. J. McCluskey. Built-in self-test for sequential machines. In *Proceedings of International Test Conference*, pages 334–341, 1987.

19. L. T. Yang and J. Muzio. A BIST testability metric-based algorithm to integrate scheduling and allocation in high-level test synthesis. In *Proceedings of the 9th International Symposium on Integrated Circuits, Devices and Systems (ISIC-01)*, pages 409–413, Singapore, September 3-5, 2001.
20. L. T. Yang and J. Muzio. Built-in self-testable data path synthesis. In Smailagic A. and De Man H., editors, *Proceedings of the 2001 IEEE Computer Society Workshop on VLSI (WVLSI-01)*, pages 78–84, Orlando, Florida, April 19-20, 2001.
21. L. T. Yang and J. Muzio. High-level data path synthesis for built-in self-testable designs. In *Proceedings of the IEEE Pacific Rim Conference on Communication, Computers and Signal Processing (PARCIM-01)*, volume 1, pages 279–282, Victoria, Canada, August 26-28, 2001.
22. L. T. Yang and J. Muzio. An improved high-level built-in self-test synthesis algorithm. In *Proceedings of the 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS-01)*, volume 1, pages 549–552, Msida, Malta, September 2-5, 2001.
23. L. T. Yang and J. Muzio. An improved register transfer level built-in self-test partitioning. In *Proceedings of the 9th International Symposium on Integrated Circuits, Devices and Systems (ISIC-01)*, pages 414–417, Singapore, September 3-5, 2001.
24. L. T. Yang and J. Muzio. An integrated high-level test synthesis algorithm for built-in self-testable designs. In *Proceedings of the XIV International Symposium on Integrated Circuits and System Designs (SBCCI-01)*, pages 115–121, Brasilia, Brazil, September 10-15, 2001.
25. L. T. Yang and J. Muzio. A register-transfer level BIST partitioning approach for ASIC designs. In *Proceedings of the 2001 IEEE Pacific Rim Conference on Communication, Computers and Signal Processing (PARCIM-01)*, volume 1, pages 275–278, Victoria, Canada, August 26-28, 2001.
26. L. T. Yang and J. Muzio. An improved BIST testability metric-based high-level test synthesis approach. In *Proceedings of the 2002 International Conference on VLSI (VLSI-02)*, pages 78–85, Las Vegas, USA, June 24-27, 2002.
27. L. T. Yang and J. Muzio. Introducing redundant transformations for built-in self-testable data path allocation. In *Proceedings of the 2002 IEEE International Conference on Communications, Circuits and Systems (ICCCAS-02)*, volume 2, pages 1346–1350, Chengdu, China, June 29-July 1, 2002.
28. L. T. Yang and J. Muzio. Introducing redundant transformations for high-level built-in self-testable synthesis. In *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems (ICECS-02)*, volume 2, pages 475–479, Dubrovnik, Croatia, September 15-18, 2002.
29. L. T. Yang and J. Muzio. Redundant transformations for the testability metrics-based built-in self-testable data path allocation. In *Proceedings of the 2002 IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS-02)*, volume 2, pages 119–123, Bali, Indonesia, October 28-31, 2002.
30. L. T. Yang and J. Muzio. Redundant transformations for the testability metrics-based high-level built-in self-testable synthesis. In *Proceedings of the XVII International Conference on Design of Circuits and Integrated Systems (DCIS-02)*, Santander, Spain, November 19-22, 2002.
31. T. Yang and Z. Peng. Register-transfer level testability analysis and improvement with pseudorandom BIST. In *Proceedings of the 1st IEEE International Workshop on Design, Test and Application (WDTA-98)*, pages 117–120, Dubrovnik, Croatia, June 8-10, 1998.

China Putian Promote Commercial TD-SCDMA Services

Qingfang Chen

CHINA PUTIAN Institute of Technology
The 2nd Street, Shangdi Information Industry Base Haidian District, Beijing, 100085 China
cqf@rd.china-putian.com

Abstract. TD-SCDMA is a 3G standard and its IPRs are owned by China which eventually will be one of the world's major Third Generation (3G) mobile markets. As the largest telecom manufacturer in China, China Putian has played an important role in the research and development of domestic TD-SCDMA sector. The industrialization of TD-SCDMA is driving along a fast lane, China Putian and TD-SCDMA Industry Alliance will try our best to make TD-SCDMA be one successful commercial system as planned.

China Putian Profile

China added 55.08 million mobile phone users in the first 10 months of this year, bringing the total number of subscribers to 325.03 million by the end of October. Eventually China will be one of the world's major Third Generation (3G) mobile markets，and this is an great opportunity for all telecom manufacturers.

As we know, China Putian is the largest telecom manufacturer in China with 7 wholly-owned manufacturing facilities, 28 subsidiaries, 50 joint ventures and 40 alliances. The company was founded in 1980. It is a key enterprise directly under the leadership of the State-owned Assets Supervision and Administration Commission of the State Council (SASAC). Its available communication equipment and terminal products mainly include: mobile communication network equipments and handsets, optical transmission equipment, microwave communication equipment, videophones and IC card payphones, PHS handsets, etc.

Headquartered in Beijing, China Putian employs 50,000 people and has revenues of 64 billion RMB ($US 7.8B) for 2003.

China Putian ranked consecutively No. 1 among Top 100 Chinese Electronic & Information Enterprises in 2001 and 2002. In 2003, China Putian ranks No. 1 among the overall listing of China's largest enterprise groups in the manufacture sector of electronic and communication equipment. It ranks No. 5 both in the listing of the 500 largest import & export enterprises and in the listing of the 200 largest export enterprises.

China Putian Institute of Technology is funded directly by China Putian Corporation in 2002, with focus on the R&D works and the all-around management for new technologies and new products.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 25-29, 2005.

China Putian has been a front-runner in the development and commercialization of TD-SCDMA technology and products. China Putian has a directorship in and is the incumbent chairman by rotation in China's TD-SCDMA Industry Alliance.

Introduction of TD-SCDMA

TD-SCDMA, or "Time Division Synchronous Code Division Multiple Access," is China's contribution to the ITU's IMT-2000 specification for 3G wireless mobile services. The dominant IPR holder behind the TD-SCDMA standard is Datang.

As one of three popular 3G standards, when TD-SCDMA is designed, the compatibility is considered, besides the advanced smart antenna, joint detection and synchronous CDMA techniques are adopted; the network structure is kept the same as 3GPP. So its outstanding advantages are shown in large system capacity, high spectrum efficiency, high ability to mitigate interference and low cost, flexibility and applicability; the core network can be shared with other systems, for example WCDMA. It can construct a network independently or construct a network with other mobile networks.

According to China's 3G planning, TD-SCDMA technology shall achieve commercialization by June of 2005. Assuming a theoretical commercial launch of all three 3G technologies by the end of next year, TD-SCDMA could account for 15% of the 3G market.

Fig. 1. Dr. Tao Xiongqiang - Chairman of the TD-SCDMA Industry Alliance

The TD-SCDMA standard is promoted by the TD-SCDMA Forum, an industry group founded in late 2002, the target of TD-SCDMA Forum is to promote the industrialization, commercialization and internationalization of TD-SCDMA. Now Dr. Tao Xiongqiang is chairman of the TD-SCDMA Industry Alliance, and he is Vice President of China Putian Corporation and President of the China Putian Institute of Technology.

TD-SCDMA has got strong supports from Chinese government. Simulative 3G-launch Project has been started by CATT of MII which is supported by government with a investment of RMB 1986 millions Yuans. It also is used for TD-SCDMA R&D and industrialization.

Vice- Chairman Mr. Zeng Qinghong, Primer Mr. Wen Jiabao, vice-Premier Mr. Huang Ju, Commissary of State Ms. Chen Zhili, Pre vice-Chairman of NPC Standing Committee Mr. Zou Jiahua and Mister of MII Mr. Wang Xudong visited TD-SCDMA booth in "PT/EXPO COMM CHINA 2004" exhibition on Oct. 27th. 2004. National leaders give a positive evaluation to TD-SCDMA fast development.

Fig. 2. Strong supports from Chinese government

The past year has seen a rapid development of the TD-SCDMA value chain, but in China's 3G planning, TD-SCDMA technology shall achieve commercialization by June of 2005. So the coming months will mark a crucial period for TD-SCDMA and will decided whether the standard achieves success.

The industrial chain for the TD-SCDMA standard has already been formed including system networks, chips and handsets. Telecom equipment providers such as China Putian, Datang Mobile, ZTE Corporation and Huawei Technologies are able to construct the TD-SCDMA networks. Many members of the alliance are working on chipsets such as T3G, Commit, Chongqing Chongyou Information Technology Co Ltd (CCIT), and Spreadtrum. More and more handset makers are manufacturing handsets supporting TD-SCDMA system, such as China Putian, Lenovo, Huawei, Amoi, DBTEL and Quanta.

Test network for TD-SCDMA is to be jointly constructed by four manufacturers: China Putian, ZTE, Datang and Nortel Networks. China Putian will work with Nortel to build a network for TD-SCDMAMTNET, a core net for the TD-SCDMA technology, in Beijing. Datang will join forces with ZTE to build a similar net in Shanghai.

China Putian's TD-SCDMA Strategy and Progess

As the chairman in China's TD-SCDMA Industry Alliance, China Putian has put heavily investment and efforts in R&D for TD-SCDMA. The R&D team of China Putian comes from China Putian Institute of Technology, which now has more than 300 full time TD-SCDMA R&D engineers.

China Putian will provide end-to-end TD-SCDMA solution: system (own UTRAN, cooperation with Nortel on core network), chipset (cooperation with COMMIT), terminal (cooperation with Bird), network planning and mobile applications. During "PT/EXPO COMM CHINA 2004", China Putian exhibits our TD-SCDMA solution. In 2005, China Putian will provide pre-commercial products that can be used in field trials in Beijing with China Telecom and China Satcom.

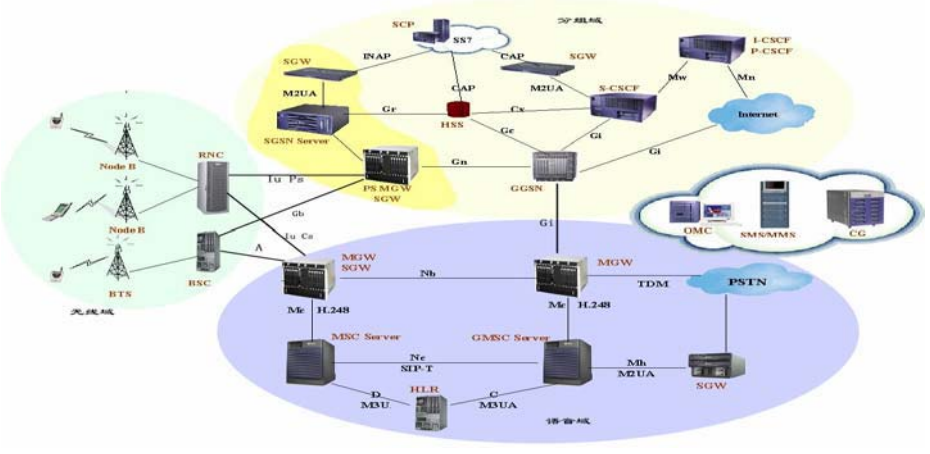


Fig. 3. Infrastructure of TD-SCDMA Total Solution

About TD-SCDMA terminal, Bird has involved in TD-SCDMA terminal activities. Totaling 230 million RMB in capital investment, COMMIT is funded by China Putian and other five globally acknowledged companies, and is developing a complete TD-SCDMA terminal chipset solution which includes: Digital Base Band (DBB), Analogue Base Band (ABB), Radio Frequency (RF), and SW protocol stack. COMMIT's TD-SCDMA terminal chipset will be launched at the end of this year.

And China Putian involved in developing two important national terminal technology standards: China Mobile Storage Standard(CMSS), Mobile Multimedia Technology Alliance(MMTA) and China Mobile Software Alliance Operating System API Standard.

Fig. 4. China Mobile Storage Standard

Conclusion

2004 is the most important year in TD-SCDMA developing history. In this year, TD-SCDMA industrialization has made great progress. Government officers, experts and carriers give a high evaluation to TD-SCDMA progress with one voice. The industrialization of TD-SCDMA is driving along a fast lane, China Putian and TD-SCDMA Industry Alliance will try our best to make TD-SCDMA be one successful commercial system as planned.

Agent-Oriented Approach to Ubiquitous Computing

Makoto Amamiya

Faculty of Information Science and Electrical Engineering
Kyushu University
Kasuga, Fukuoka, Japan 816-8580
amamiya@is.kyushu-u.ac.jp

Abstract. With today's developments in device miniaturization, wireless networking, such as PCs, PDAs, Cell phones, RFID tags and so on, we are facing great opportunities and challenges to realize the *ubiquitous computing* vision. In this paper, we first identify the key characteristics of ubiquitous computing systems; then argue that agents and agent networks are the right metaphor for managing the dynamism and complexity of system integration and on demand interactions in ubiquitous computing systems.

1 Introduction

As the 20th century turned into the 21st century, hardware technologies are further advanced by device miniaturization, wireless networking, mobile devices and many others. With this background, the new challenges for software technologies are the realization and exploitation of the ubiquitous computing vision of "*the network and computer are everywhere*," which is beyond the scope of Web technologies.

The drawback of Web technologies is that they are typically described in terms of protocols on top of the TCP/IP stack. Some well-known examples include World Wide Web (HTTP), electronic mail (SMTP), file transfer (FTP), virtual terminal (TELNET), etc. Thus different applications use different protocols and interoperability between them is hard, if not impossible. Even worse, those tailor-made protocols are hard to be enhanced or replaced because they require worldwide changing of independently written implementations. Defining protocols and testing their compatibility on different system platforms are no longer rational and feasible for ubiquitous computing systems in which on-demand integration and interactions are involved all the time.

Our solution lies in the development of a *global computing architecture* based on agent-oriented programming paradigm. Rather than developing applications directly in terms of protocols, agents are basic building blocks from which diverse application systems can be abstracted, organized and constructed. This paradigm, compared with more traditional approaches, can be best characterized by autonomous software agents, agent organizations and agent interactions.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 30–37, 2005.

Through our work, we concentrate on enabling technologies for building a scalable, flexible and secure agent networks that provides us with a convenient programming model, sufficient transparency and interoperability.

The argument presented in this paper examines the new challenges in ubiquitous computing, and concentrates on the approaches we take to formalize the agent-based global software infrastructure.

2 New Challenges

We base our analysis of ubiquitous computing, or ubicomp, on common scenarios that Mark Weiser [1] and others presented [2,3,4,5] as the followings:

Specialized elements of hardware and software artifacts — connected by wired or wireless networks — interact to support human-centered activity seamlessly.

The underlying challenges of ubiquitous computing software, therefore come in seven parts [6]:

- *integration*: Due to the inherent openness, components of ubicomp systems changes constantly and it is thus impossible to have prior knowledge of their ultimate size. They should be dynamically configured and integrated on demand.
- *self-adaptation*: Some computing contexts of ubicomp vary physically — such as device characteristics, resources and access methods [7]. Others vary logically — such as personal preferences, current session state, and history of interactions. This requires the capability of supporting self-adaptation.
- *discovery*: When a component enters an environment, mutual discovery takes place between it and other available services and devices, and find out to whom the interaction is appropriate.
- *location-aware*: Ubicomp systems are usually divided into environments with boundaries, which often, but not necessarily, specified by their physical location, such as homes, offices, or museums.
- *interaction*: In a ubiquitous system, components must spontaneously interact with each other to make their resources and services accessible in changing environments because they all vary unpredictably from moment to moment.
- *robustness*: Compared to wired distributed systems, ubiquitous systems are supposed to face much more transient failures, especially those in wireless networking. When failure is a common case, system designers should clearly indicate which operations are failure-free or failure-prone.
- *security*: Security is so critical in open environments that no one can ignore it. Our experiences also confirm that security issues, such as the protection of components' resources and privacy, trust and authentication, system availability, data confidentiality, and data integrity, should be carefully considered at the stage of design.

3 Agent-Based Approach

Nowadays, *software agent* (or *agent* for short) has been advocated by many researchers and software developers as a promising and innovative way to model, design and implement complex distributed systems [8,9]. The key notion behind this new paradigm is a *self-directed behavioral structure* [10]. In effect, it is a new programming tool that emphasizes the idea of interaction, as well as the idea of choice and options at the time of action, rather than at the time of programming.

On analogy with object-based concurrent programming (OBCP), our approach is based on *agent-based concurrent programming* (ABCP). Basically, agent-based systems are more difficult to correctly design and implement than other non-agent systems. This is mainly because agents are computational entities that perceive their environment through sensors and act upon their environment through efforts. In this diagram, three aspects of an agent are identified.

- An agent has to have a repertoire of possible actions available to it which constitute its ability to modify its environments.
- An agent has to have some interaction with the world around it and get feedback about its choice, whether it is successful or not. Interaction can take place indirectly through the environment (e.g., by carrying out an action that modifies the environmental state) or directly (e.g., by exchanging information with other agents) through a shared language.
- An agent has to have a continuous operating engine that persistently strives for success without any need to account at the outset for all possible conditions it will face or in what order it will face them. What programmers provide to an agent is a goal or a utility function and a collection of building blocks for getting there and, in some advanced cases, even a way to learn something new that is needed.

3.1 Layered Architecture

In an instance of the idea that "the network and computer are everywhere," agent technology emphasizes the ability to reach out, discover and interact with others.

Table 1. Examples of logic at each layer

application	problem solving
	choice and options
	self-learning
agent	interaction pattern
	social relationship
	name addressing (agent world)
	security policy (agent world)
network	agent name resolution (network world)
	agent message deliver
	quality of service management
	network security (network world)

Such an emphasis raises a number of challenging issues that all are centered around an elementary question of *what*, *when* and *how* to interact with *whom*.

Through our work on the KODAMA (Kyushu University Open & Distributed Autonomous MultiAgent) project [11], we have established a *separation* principle that mandates the separation of application-level logic from agent-level logic and the separation of agent-level logic from network-level logic. Some examples of logic at different layers are given in Table 1.

In accordance with the separation principle, a layered architecture has been adopted. This layered architecture makes higher-level (i.e. application-level and agent-level) logic to deal with the problem of *what*, *when* and with *whom* to interact, while low-level (i.e. network-level) logic concentrates on the problem of *how* to interact.

By detaching data exchange activities from agent programmes, it is possible to build generic agent communication facilities in the network layer. This helps improve productivity in several ways.

- First, agent programmers can focus on higher-level abstractions only, and leave the low-level details of communication to network layer programmers. They do not need to write special software to move data between each possible pair of agents.
- Second, agent development is not restricted to a particular architecture, thus keeping the entire system flexible. Agents, for example, can remain unchanged while the network layer are reconfigured or updated and vice versa.
- Third, low-level logic handles agent communication traffic without understanding the applications that use it. The specification of the network layer can be publicly available and it accommodates a wide variety of underlying hardware, communication technologies and contents of agent communication.

It is important to note that both agent-layer logic and network-layer logic are application independent. The main purpose of having these two levels, therefore, is to integrate a set of common distribution services that forms a uniform developing platform upon which various applications can be efficiently built with inherent support of cooperation.

Furthermore, a *plug-and-play* standard has been deployed to separate the application-level logic from the agent-level logic within an agent. More precisely, an agent is made up of a *kernel* unit, which encapsulates the common modules, and an *application unit*, which encapsulates the application-dependent modules [11]. In this way, the implementation aspects of data sharing, exchange and management among agents are made transparent to application programmers.

On the other hand, the wired or wireless network itself is full of latencies, congestion, overload and unforeseen failure. In the layered architecture, there is a specific layer, we call it *agent communication infrastructure* (or *agent infrastructure* for short), on which agents running in different physical spaces can

Table 2. Layered architecture

level	layer
application	*agent application unit*
agent	*agent kernel unit*
network	*agent infrastructure* (middleware)
	networking

Table 3. Agent/Infrastructure interface (registration and de-registration)

method	time	parameter	return value
`register`	creation, immigration	agent logical name (in agent layer)	message queue
			`false`
`remove`	deletion, emigration	agent logical name	`void`

communicate with each other easily, freely and without concern about the interconnection issues. Actually, this is another separation between agent-level logic and network-level logic.

In sum, from the highest layer to the lowest level, the layered architecture consists of *agent application unit* layer, *agent kernel unit* layer, *agent infrastructure* layer and *networking* layer. The relations between the separation principle and the layered architecture are summarized in Table 2.

3.2 Agent Layer

The *agent* layer stays on the top, which can be further divided into application unit sub-layer and kernel unit sub-layer. Typically, all agents speak one or more languages called agent communication language (ACL). ACL itself only specifies the format, or syntax, of the information being transfered. The meaning, or semantics of the information on the other hand, is specified by application logic. Such considerations are also reflected in our plug-and-play architecture of agents. That is, the basic support of ACL, such as message validating and message parsing is integrated into the kernel unit. On the other hand, message contents and message interpretation are open to different application units. As a consequence, once the kernel unit is finished, it can be used by all agents, whereas various application units need to be developed to meet the exact requirements of various applications.

Table 4. Agent/Infrastructure interface (agent message delivery)

method	time	parameter	return value
`put`	send message	agent message	`void`
`get`	receive message	`void`	agent message

However, agent interaction is more complex than interactions in conventional models, such as client-server or publish-subscribe. This is because data is transmitted among agents (sometimes users), regardless of whether a prior relationship exists. At the time of design, for example, nobody knows how many agents will be created, where agents will reside, or what agents will do. Rather, in a practical, worldwide, distributed agent-based system, interaction may occur at unpredictable times, for unpredictable reasons, between unpredictable components [12].

To support the scenario of agent communication, all agents are usually logically organized and located into various *agent communities*, which in turn may be linked together to form a unified agent society. As a consequence, an agent system can be divided into a number of top-level communities and each one may cover many agents or be partitioned into sub-communities. In this way, any agent can be uniquely located and named by giving its community position. It is also possible for agents to join two or more communities concurrently, so that they can be found in different communities. Agent communities have a dynamic membership because agents can join or quit from time to time. Social networks of agents, as a whole, can develop in an evolutionary fashion.

3.3 Agent Infrastructure Layer

Next comes the *agent infrastructure* layer, which plays a vital role in connecting the agent layer with the networking layer. Such connection is guaranteed and realized by two well-defined interfaces, one between the agent layer and the agent infrastructure layer, and the other between the agent infrastructure layer and the networking layer.

Beyond physical interconnection, this layer provides additional services tailormade for agent communication over networks. Some examples of such services include *agent name resolution*, *agent message delivery*, *mobile computing support*, *quality of service management* and *information security*. Additionally, this layer absorbs the variety and complexity of communication processes, and make the collection of agent systems appear to be a single large-scale and open system.

In practice, agents are self-contained entities that rely on the agent infrastructure layer to provide transparent support for on-line interactions. Once agent messages are passed from agents to the agent infrastructure, they should be delivered to their destination without further interaction with agents. With the relationship and interaction between the two layers clear, it is natural to define the contents of their interface. Two examples are given in Table 3 and Table 4.

3.4 Networking Layer

The underlying layer below the agent infrastructure is the networking layer in which the communication service is concretely realized. Currently, the TCP/IP protocol suit is the *de facto* standard in the wired networks while wireless networking technologies include IEEE 802.11, bluetooth, infrared, etc.

In a layered system, each layer provides services to the layer above it and serves as a client to the layer below it, and interfaces between adjacent layers determine how layers will interact [13]. Usually, interfaces are well-defined so that higher layers are hidden from lower ones. This approach has several desirable properties.

- First, it supports designs based on increasing levels of abstraction.
- Second, it dramatically simplifies system enhancement and maintenance. Changes to the function or interface of one layer, for example, affect at most two other layers (below and above).
- Third, it supports component reuse. Whenever the interfaces are the same, different implementations can be built and used interchangeably.

4 Conclusions

This paper addressed the importance of the agent-based approach to ubiquitous computing, and gave a general introduction to an agent-based software architecture. In particular, the discussion was focused on the layered architecture which is configured with *agent application unit* layer, *agent kernel unit* layer, *agent infrastructure* layer and *networking* layer.

We have conducted several work on applications (see [14,15,16,17]), and we are convinced that agent-based approaches to information management are both scalable and cost-effective for real-world ubiquitous applications. However, our work is still relatively simple and has some limitations. Instead of in-house program development, for example, a formal specification of agent roles and interaction protocols is needed to close the gap between analysis phase and subsequent development and verification phases. Another important issue for real-world applications is security management in message communication between agents.

References

1. M. Weiser. The computer for the 21st Century. Scientific American, Vol. 265, no. 3, pages 94–104, September 1991.
2. M. Fleck, M. Frid, T. Kindberg, E. O'Brien-Strain, R. Rajani, and M. Spasojevic. From Informing to Remembering: Ubiquitous Systems in Interactive Museums. IEEE Pervasive Computing, 1(2):13–21, April–June 2002.
3. V. Stanford. Using Pervasive Computing to Deliver Elder Care. IEEE Pervasive Computing, 1(1):10–13, January–March 2002.
4. M. Beigl, H. W. Gellerson, and A. Schmidt. MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects. Computer Networks, 35(4):401–409, March 2001.
5. Joseph M. Kahn, R. H. Katz, and Kristofer S. J. Pister. Mobile Networking for Smart Dust. In *Proc. of Int'l Conf. Mobile Computing and Networking*, pages 271–278, ACM Press, 1999.
6. T. Kindberg, and A. Fox. System Software for Ubiquitous Computing. IEEE Pervasive Computing, 1(1):70–79, January–March 2002.

7. P. Bellavista, A. Corradi, and C. Stefanelli. The Ubiquitous Provisioning of Internet Services to Portable Devices. IEEE Pervasive Computing, 1(3):81–87, July–September 2002.
8. G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, USA, 2000.
9. F. Zambonelli, N. R. Jennings, A. Omicini, and M. Wooldridge. Agent-oriented software engineering for Internet applications. In *Coordination of Internet Agents*, pages 326–346. Springer-Verlag, New York, USA, 2001.
10. L. Gasser. Agents and concurrent objects. *IEEE Concurrency*, 6(4):74–77, 81, October-December 1998. Interviewed by Jean-Pierre Briot.
11. G. Zhong, S. Amamiya, K. Takahashi, T. Mine, and M. Amamiya. The design and implementation of KODAMA system. *IEICE Transactions on Information and Systems*, E85-D(4):pp. 637–646, April 2002.
12. N. R. Jennings. Agent-based computing: Promise and perils. In *Proc. of Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1429–1436, July 1999.
13. E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
14. B. Hodjat and M. Amamiya. Introducing the Adaptive Agent Oriented Software Architecture and its Application in Natural Language User Interfaces. In *The First Workshop on Agent Oriented Sftware Engineering (AOSE-2000), Lecture Notes in Computer Science vol.1957*, Springer-Verlag, pp.285-306, 2001.
15. T. Helmy, S. Amamiya and M. Amamiya. Collaborative Kodama Agents with Automated Learning and Adapting for Personalized Web Searching. In *Proceedings of the Thirteenth Innovative Applications of Artificial Intelligence Conference (IJCAI/IAAI-2001), Emerging Application, Technology, and Issue Paper*, Seattle, Washington, USA, pp. 65–72, 2001.
16. K. Takahashi, S. Amamiya, T. Iwao, G. Zhong, and M. Amamiya. An Agent-based Framework for Ubiquitous Systems. In *Proc. of the 2nd International Workshop on Challenges in Open Agent Cities*, pp. 49–52, July. 2003.
17. T. Iwao, S. Amamiya, K. Takahashi, G. Zhong, T. Kainuma, L. Ji and M. Amamiya. Information Notification Model with VPC on KODAMA in an Ubiquitous Computing Environment and its Experiment. In *CIA2003, LNAI 2782*, pp. 30-45, Aug. 2003.

Resource-Aware Programming*
Invited Paper

Walid Taha

Rice University, Houston, TX, USA
taha@rice.edu

Abstract. Traditional wisdom in programming language design suggests that there is a trade-off between expressive power and static guarantees. We describe a novel schema for designing a class of languages that we call Resource-aware Programming (RAP) languages. By taking into account the natural distinction between the development platform and the deployment platform for embedded software, RAP languages can alleviate the need for drastic trade-offs between expressive power and static guarantees. We describe our preliminary experience designing and programming in a RAP language for hardware design, and give a brief overview of directions for future work.

1 Introduction

Designers of embedded and real-time software must attend not only to functional specifications, but also to a wider range of concerns, including resource consumption and integration with the physical world. In current practice, the dominant medium for programming is various dialects of C. This is a puzzling state of affairs, given that: First, C is now over thirty years old, has many well-known limitations, including several well-known safety problems, and has a limited set of abstraction mechanisms; second, since then the programming languages community has produced new languages that address many of these safety problems, and developed several powerful abstraction mechanisms. Today, there is pressing need for addressing this issue. In particular, as new embedded hardware platforms continue to flow into the embedded systems market, the need for effective techniques for producing reliable software in a cost-effective manner becomes more pressing.

Real, technical challenges have hampered the adoption of language innovations in the embedded software domain. Possibly the most important reason is that it often appears as if a choice has to be made between expressive power and static guarantees. Expressive programming languages, often referred to as high-level languages, generally offer powerful abstraction mechanisms like higher-order functions, managed dynamic data structures, and general recursion. At the

* Supported by NSF ITR-0113569 "Putting Multi-stage Annotations to Work" and Texas ATP 003604-0032-2003 "Advanced Languages Techniques for Device Drivers."

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 38–43, 2005.

same time, these languages provide little or no guarantees about resource utilization. Because such languages tend to deprive the programmer of control over resources that she *must* take full responsibility for, they are generally not well-suited for building embedded applications. Resource-bounded languages, such as state charts or synchronous languages, provide strong guarantees about the runtime behavior of programs. Because such languages generally deprive the programmer of constructs that allow her to write concise, structured, modular, and reusable programs, an unsafe language that provides more expressive power can be significantly more attractive in practice.

While there are several important innovations as well as ongoing efforts in the programming languages community to offer better trade-offs between these poles, an important insight has long been overlooked. Our key observation is that this apparent need to choose between expressive power and static guarantees can be often be avoided. *Resource-aware Programming (RAP) languages* are a class of languages aimed at addressing the problems described above by:

1. Providing a highly expressive *untyped substrate* supporting state-of-the-art abstraction mechanisms such as dynamic data-structures, modules, objects, and higher-order functions. The role of this substrate is to provide a common, unified model of the semantics of the whole computation, starting from what happens on the platform used to design the software, and extended to what must take place on the embedded platform where the software must ultimately operate. Because of their simpler reasoning principles and the wealth of results on statically checking them, our studies tend to use functional programming languages as untyped substrate [5]. In principle, these ideas should be applicable to any programming language.
2. Allowing the programmer to express the *stage distinction* between computation on the development platform and computation on the deployment platform. Expressing the stage distinction is, in principle, achieved by any language that can support program generation or that has a macro-expansion facility. But mechanisms based on strings or s-expressions would be insufficient, as they would interfere with the possibility of automatic static checking of programs *before* they are generated.
3. Using *static checking* to ensure that computations intended for execution on resource-bounded platforms are indeed resource-bounded. In fact, the ability to perform this kind of static checking is the most novel feature of RAP languages. To get an appreciation for the importance and the challenge involved in doing this, consider the analogous situation in the context of C: It would correspond to statically checking the safety and resource usage of C programs *before* they are pre-processed using configuration parameters for various target platforms.

The combination of these three ingredients allows the programmer to use sophisticated abstraction mechanisms in programs that are statically guaranteed to generate only resource-bounded programs. We expect that languages with these features can provide a solid bridge between traditional software engineer-

ing techniques on one side, and the specific demands of the embedded software domain on the other.

For general-purpose programming, the idea of statically checked generators has been studied extensively, largely in the context of *multi-stage programming* [3]. For general-purpose software, statically checked generators provide a mechanism for avoiding the runtime overhead typically associated with abstraction mechanisms such as functions and objects. For embedded software, the primary role of such generators will be to allow powerful abstraction mechanism to coexist with statically checkable properties on resource usage.

To date, our preliminary efforts to explore the idea of RAP languages have consisted of two main efforts: First, we have shown how a heap-bounded programming language can be extended with higher-order features [4]. Our experience in this study suggests that the static checking problems that arise in designing a RAP language can be non-trivial but nevertheless tractable. Second, we have shown how to use a two-stage language to concisely express Cooley and Tukey's recurrence that defines the Fast Fourier Transform (FFT) [1,2]. These definitions are essentially program generators which can be used to generate exactly the butterfly circuit for FFT for any size 2^n. Our experience with this effort is discussed in the following section.

2 A RAP Hardware Description Language

RAP languages can play an important role in hardware design because, except for very high-end applications, verifying the correctness of hardware systems can be prohibitively expensive. In contrast, software languages are primarily concerned with issues of expressive power, safety, clarity, and maintainability. Software languages can provide abstraction mechanisms, which make designs more maintainable and reusable. They can also keep programs close to the mathematical definitions of the algorithms they implement, which helps with ensuring correctness. Hardware description languages such as VHDL and Verilog provide only limited support for such abstraction mechanisms. A RAP language for hardware circuits would allow us to capture the schema (or generator) for a family of circuits in an executable form. With such a schema, rather than having to verify circuits on a case-by-case basis, a unified substrate for the full process would enable the verification of a whole *family* of circuits en bloc.

A basic method for building a circuit schema in a RAP language has been proposed [1]. In addition to allowing us to implement a schema for FFT circuits concisely, following this systematic approach also yielded new insights into the relation between the FFTW and Split-radix implementations [2]. In this method, we start with naively-generated circuits that are correct by construction. In the case of FFT, this becomes evident because the schema is almost a literal transliteration of a textbook definition of the recurrence defining FFT. Then, more efficient circuits are correct as long as they are produced by systematic, verified improvements on a correct but naive generator. Note that these improvements can be carried out by improvements on the schema. Note also

that correctness is *not* achieved by verifying a naive generator and verifying a posteriori (post-generation) optimizations that fix up the result of the generator. This means that we replace the problem of verifying transformations to one of verifying modifications to one program: the generator.

2.1 Manifest Interfaces, Composition and Static Checking

As noted briefly in the introduction, statically checking generators can be hard to achieve using traditional type systems. For example, if strings, algebraic datatypes, parse trees, or even graphs are used to represent the generated program, they would only allow us to express a manifest interface with a type such as: `gen_fft : int -> circuit`, where `circuit` is the type we choose to represent circuits with. The static type `int -> circuit` says that `gen_fft` is a function that can only take an integer and can only produce a circuit. As soon as we start *composing* generators — for example, if we want to build a circuit that computes the FFT, performs a multiplication, and then computes the inverse FFT — we run into a problem: The type `circuit` does not provide any static information or guarantees about the consistency or well-formedness of the composite circuit. This is an instance of a general need for *manifest interfaces* that would provide us with enough static information to allow us to guarantee some degree of well-formedness on the result of the composite program. To illustrate, assume we are given two trivial generators which take no inputs and produce an AND-gate and an inverter:

```
and : circuit
inv : circuit
```

A meaningless composition arises if we write `let bad = inv --> and`, where the connect operator `-->` is an infix operator that has the type

```
circuit $\times$ circuit -> circuit
```

and which wires the output of its first circuit to the input of the second circuit. The problem is that the second circuit does not have just one input but *two*, and the type system does not prevent this error: All circuits just have type `circuit`.

It is generally desirable that the `circuit` type be as expressive as possible, but at the same time only express values that are circuit-realizable. For example, the programmer might want to use abstractions such as lists (or any other dynamic data structure) in describing the circuit, but will need to know as early as possible in the development process that these uses can be realized using finite memory [4].

2.2 Better Static Checking

Rather than using one concrete type to represent circuits, a RAP language provides an abstract datatype parameterized by information about the generated circuit. The type of the two trivial generators above would be:

```
and : (bool × bool -> bool) circuit
inv : (bool -> bool) circuit
```

The type of the connect operator `-->` would be refined from being

```
circuit × circuit -> circuit
```

to being

```
(α -> β) circuit × (β -> γ) circuit -> (α -> γ) circuit
```

where α, β, and γ are generic type variables that must always be instantiated consistently. With this extra information, the type system can reject the above `bad` declaration, because the type variable β cannot be instantiated to both the output of `inv` (which is `bool`) and the input of `and` (which is `bool`×`bool`). Note that the type of this function is similar to the type of the standard mathematical function composition operation of type `(α -> β) × (β -> γ) -> (α -> γ)`.

2.3 Safe Implementations of Domain-Specific Optimization

To ensure that generated programs are well-typed and resource-bounded *before* they are generated, the circuit type constructor in a RAP language must remain abstract, meaning, that there is no mechanism within the language to allow the programmer to de-construct code once it has been generated. Providing constructs for traversing values of this type jeopardizes the soundness and decidability of static typing, and complicates reasoning about the correctness of programs written in these languages. At the same time, not being able to look inside the generated circuit descriptions means that a posteriori optimizations cannot be expressed within the language. While such optimizations can still be implemented as stand-alone source-to-source transformations *outside* the language, doing so invalidates the safety and resource-boundedness guarantees.

We distinguish two forms of a posteriori optimizations: Generic ones that are independent of the application, and ones that are specific to the application. Generic optimizations are generally well-tested and are less likely to invalidate the guarantees provided by the RAP setting. Such optimizations can be provided as special extensions of the language as long as they have been proven to preserve all guarantees. But domain-specific optimizations written by the programmer for a particular application are less likely to have been tested as extensively, and are therefore more problematic. At the same time, systems such as FFTW make a strong case for the practical importance of such domain-specific optimizations.

We were able to show that *abstract interpretation* on program generators can be used to avoid the need for a posteriori optimization [1]. This allows us to generate the desired circuits without losing the guarantees provided by RAP languages. The benefits of the proposed technique extend to the untyped setting, as it avoids the generation of large circuits in the first place, thus reducing the overall runtime needed to generate acceptable circuits. From the verification point of view, this approach replaces the problem of verifying a source-to-source transformation to that of verifying the correctness of a finite set of optimizations on *one* specific program: the generator.

3 Key Directions for RAP Research

The design space for RAP languages is huge, primarily because there are numerous notions of resource-boundedness and languages that can be considered for the deployment platform, as well as the numerous abstraction mechanisms that may be desirable on the development platform. A systematic survey is therefore beyond the scope of this paper. However, there are a number of broad directions that we expect to be important to progress in this area:

- Extensions of traditional static analysis techniques (including type systems) to work in a generative setting. We expect our own efforts to focus on analysis that have direct applications in challenging domains, such as device drivers, control systems, and hardware description languages.
- Better understanding of the process of writing RAP programs, including further study of the use of program structuring mechanisms such as monads, as well as the use of abstract interpretation as a programming technique for implementing domain specific optimizations.
- Support for certification. In particular, while previous work on RAP languages have so far focused on static guarantees, the execution model on the development platform can be naturally extended to preserve the proof behind this guarantee, and this proof can then be produced along side the deployment platform computation. Such a certificate can be verified independently of the generation process, much in the same way as proof-carrying-code is used to verify the safety of software received from an untrusted source.

Acknowledgment: Anthony Castanares, Emir Pašalić and Kedar Swadi have kindly read and commented on drafts of this paper.

References

1. Oleg Kiselyov, Kedar Swadi, and Walid Taha. A methodology for generating verified combinatorial circuits. In *the International Workshop on Embedded Software (EMSOFT '04)*, Lecture Notes in Computer Science, Pisa, Italy, 2004. ACM.
2. Oleg Kiselyov and Walid Taha. Relating FFTW and Split-Radix. In *Proceedings of the International Conference on Embedded Software and Systems*, 2004. Appears in this volume.
3. Walid Taha. A gentle introduction to multi-stage programming. In Don Batory, Charles Consel, Christian Lengauer, and Martin Odersky, editors, *Domain-specific Program Generation*, LNCS. 2004.
4. Walid Taha, Stephan Ellner, and Hongwei Xi. Generating Imperative, Heap-Bounded Programs in a Functional Setting. In *Proceedings of the Third International Conference on Embedded Software*, Philadelphia, PA, October 2003.
5. Walid Taha, Paul Hudak, and Zhanyong Wan. Directions in functional programming for real(-time) applications. In *the International Workshop on Embedded Software (EMSOFT '01)*, volume 2221 of *Lecture Notes in Computer Science*, pages 185–203, Lake Tahoe, 2001. Springer-Verlag.

In-House Tools for Low-Power Embedded Systems

Naehyuck Chang

School of CSE, Seoul National University, Korea
naehyuck@snu.ac.kr

Abstract. Power consumption emerged as a distinct axis for system optimization especially for battery operated applications. Most of all, circuit and device level low-power design has leveraged battery-operated embedded systems over dozens of years. As of today, high-level or system-level power reduction is believed for another significant power saving opportunity. Nevertheless, existing power-related tools are not familiar with system and software designers, who have to pay more attention to power consumption than other optimization factors.

In this paper, we introduce a series of power measurement and estimation tools that differentiate the quality and effectiveness of high-level power reduction practices for embedded systems. To fulfill necessary requirement for high-level power reduction, we have developed a cycle-accurate energy measurement technique using switched capacitors. This new technique enabled us to develop innovative power measurement tools for memory devices, FPGAs and CPUs. This individual power measurement tools contribute quality energy characterization of components, and eventually come up with an integrated system-level power estimation tool: SEE (Seoul National University Energy Explorer, http://see.snu.ac.kr).

1 Introduction

Together with speed and cost, energy consumption is now a primary performance metric for battery-operated embedded systems. A well-designed embedded system should be globally optimized to the target application, from user interface right through to device technology. This kind of global optimization over many layers of software and hardware is challenging, due to the need for extensive inter-disciplinary collaborations. Energy estimation is a routine job in low-level hardware design. Unfortunately, at this stage, the specific application of most hardware components is not known, and designers cannot perform an application-specific optimization. Another opportunity for optimization is given to software and system designers; but they are often unfamiliar with hardware-related energy issues. This problem is compounded because traditional energy estimation tools like SPICE and PowerMill [1] are designed for use by low-level hardware engineers, which can discourage designers working at a higher level to attempt global optimization.

With the increasing trend towards low-power design, a higher-fidelity, system-level energy estimation environment is demanded. In this paper, we propose a series of energy measurement, estimation and exploration tools, which overcome limitations of existing tools.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 44–58, 2005.

2 Related Work

A simple approach is to analyze actual measurements from a hardware platform. Tools like Powerscope [2] and Itsy [3] use computer-controlled multi-meters or A/D converters to measure energy consumption. Other studies adopt software energy estimation methodology, which has been an important aspect of embedded systems design since it was first introduced by [4]. While the early literature [4] focuses on the possibility of software optimization, recently proposed tools extend the methodology to support high-level hardware optimization [5,6,7,8]. JouleTrack [5] is a publicly available web-based software energy profiling tool for processor cores. SimplePower [6] and Wattch [7] estimate the power consumption of processors including the on-chip cache, on-chip bus and on-chip SRAM, but still excluding off-chip subsystem. JouleTrack and similar systems [5,6,7] are good for architecture-level analysis since they only consider processors. On the other hand, a power estimation framework [8] presents a system-level energy estimation that includes a processor, L1/L2 cache, off-chip memory, and DC-DC converter. However, energy models for off-chip memory devices are too simple to support a cycle-level analysis. A simple power characterization is taken and devices are assumed to have only two modes: active and idle. Consequently, most high-level energy estimators are not incapable of cycle-accurate analysis of each important component, since energy consumption is averaged out over the entire execution time, which means that they are not suitable for high-level power reduction but for high-level power estimation.

3 Power Characterization for High-Level Power Reduction

High-level design such as RTL and behavior levels, overcome explosive complexity by proper abstraction that hides low-level design details. In that sense, high-level power saving may not have to consider microscopic power changes. This is true for high-level power estimation. Suppose we have three different characterization schemes of gas consumption of a vehicle as shown in Table 1. All the three different characteriza-

Table 1. Characterization of gas consumption of a vehicle (G, S, I, R, n, c: gas consumption, vehicle speed, idle gas consumption, engine restarting cost, number of engine starts, and a constant, respectively).

Characterization 1	Linear mode	$G = cS$
Characterization 2	Non linear model	$G = cS + I$
Characterization 3	Including restarting cost	$G = cS + I + nR$

tion schemes are useful for high-level gas consumption estimation. Of course, Characterization 3 is more accurate than Characterization 2, and Characterization 2 is better than Characterization 1. However, if a vehicle stops only a few minutes and the engine is never turned off during the entire trip, all the characterization schemes may show similar accuracy. Now, suppose we need to devise a gas consumption saving scheme that

is useful when a vehicle temporary stops at a parking space. Characterization 1 shows that the vehicle does not consume any gas when the speed is zero, *i.e.* the vehicle stops. Thus, there is even no need to devise such a gas consumption saving technique. Characterization 2 shows that I is still consumed while the vehicle stops. Thus, the best way to save gas consumption from Characterization 2 is turning off the engine whenever the vehicle stops even for just a second. Characterization 3 considers the engine restarting cost, and thus we better keep the engine running during a short stop, which is a practical solution while previous ones are not applicable to real situation. Consequently, for the derivation of a power saving policy, we must be extremely careful in abstraction of low-level behaviors even for high-level approaches, unlike in case of high-level power estimation.

4 Energy State Machine

In this paper, we introduce an *energy state machine* to describe accurate energy consumption behavior of digital systems where each node and each transition describe the static power and the dynamic energy, respectively. A finite state machine M is four-tuple (S, Σ, δ, s_0) where $S = (s_0, ..., s_n)$ is a set of finite states, Σ is a finite input alphabet, $\delta : \Sigma \times S \rightarrow S$ is a state transition function, and s_0 is the initial state. Each arc, that denotes state transition δ, is labeled with a finite set of state transitions $T = (t_0, ..., t_m)$. Fig.1 illustrates the variation of power supply current i_{dd} in asynchronous and synchronous devices. It also justifies the static and the dynamic energy association of the energy state machine. Asynchronous devices consume the dynamic energy when strobe

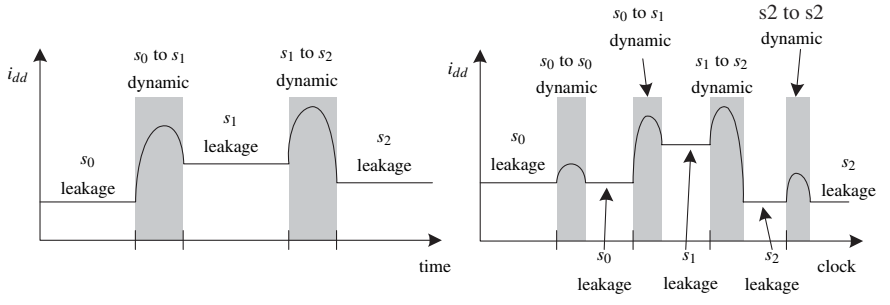
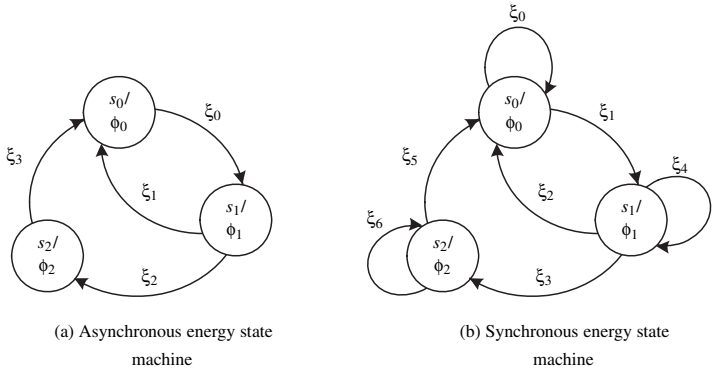


Fig. 1. i_{dd} variation due to state change.

signals are issued. The strobe signal changes the device state leading to variation of the static energy consumption. Synchronous devices consume the dynamic energy at each clock edge rather than logical state change.

Definition 1. *Energy state machine* Π *is three-tuple* (M, Φ, Ξ) *where* M *is a finite state machine,* $\Phi = (\phi_0, ..., \phi_n)$ *is leakage energy associated with state* $S = (s_0, ..., s_n)$*, and* $\Xi = (\xi_0, ..., \xi_m)$ *is the dynamic energy associated with transition* $T = (t_0, ..., t_m)$.



(a) Asynchronous energy state machine

(b) Synchronous energy state machine

Fig. 2. Energy state machine.

5 Cycle-Accurate Energy Measurement

To complete the energy state machine, we have to annotate the energy values Φ and Ξ. At first, we need to distinguish energy consumption behavior of target devices before deciding a characterization method. While synchronous energy FSM is ideal for high-fidelity characterization of energy consumption for synchronous digital systems, a special technique is required to annotate energy values for transitions and states. As shown in Fig. 1, dynamic energy consumption represented by the i_{dd} current only occurs during the propagation time, which is usually a matter of nanoseconds. The propagation delay is not determined by the operating frequency but by the physical design, and thus the power spectrum of i_{dd} reaches well over several hundred MHz, regardless of the operating frequency. This seriously discourages us from trying to distinguish i_{dd} from the cycle-by-cycle dynamic energy using conventional equipment such as an ammeter [9].

Since cycle-accurate energy measurement is essential to annotate a synchronous energy FSM, we have developed a special technique to handle the cycle-by-cycle energy measurement of high-speed digital systems [9]. Fig. 3 shows a schematic diagram of the measurement setup. We transfer charges to the capacitor and operate the target device using these charges. By simply measuring the initial and the final voltage at the capacitor, we can derive the exact energy consumed by the target device.

There are on-chip bypass capacitors for mitigating power supply fluctuation in most modern devices, which make energy calculations complex. We will denote this capacitance by C_B in Fig. 3: its value is determined by the charge-sharing rule [10]. In addition, modern high-performance devices are not generally free from leakage current. We add R_S into the device model to represent the leakage current. While most system-level energy simulators are primarily concerned about C_L, we use fairly realistic energy models for both measurement and characterization.

Fig. 4 shows a waveform captured by a DSO for demonstration purposes, using high-speed, pipelined analog-to-digital converters. Depending on the design, the voltage drop is variable. We minimize the quantization error of the analog-to-digital

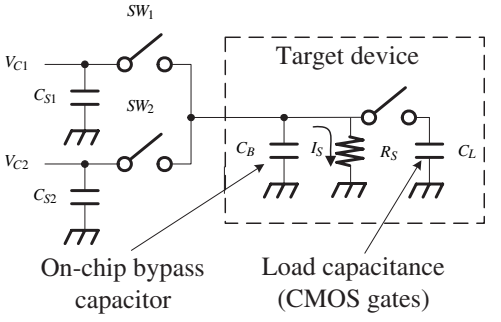


Fig. 3. Cycle-accurate energy measurement using switched capacitors.

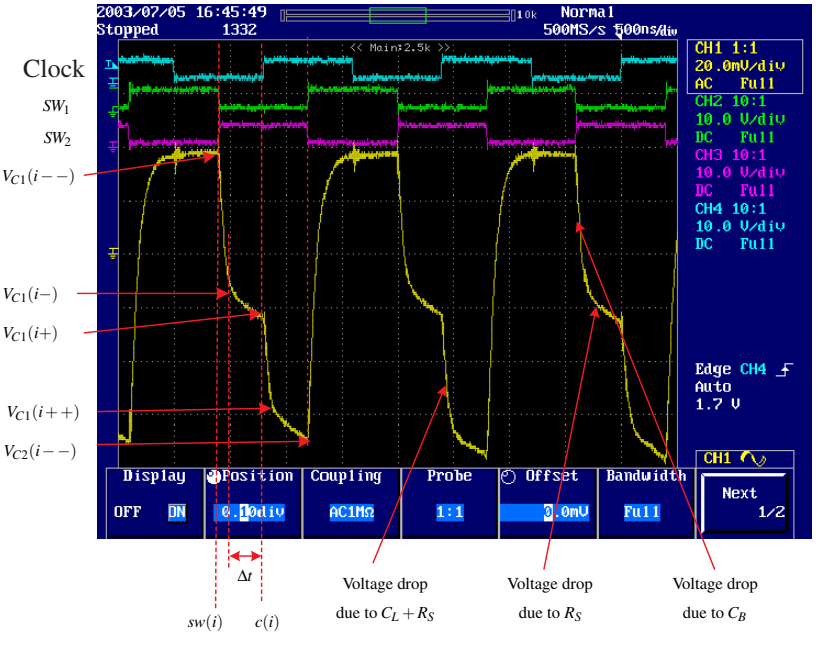


Fig. 4. Voltage across the two switched capacitors in Fig. 3.

converter by adjusting the capacitance of C_{S1} and C_{S2} to correspond to the full-scale voltage swing of the analog-to-digital converters. This explains why the measurement tool must be customized to a target device, which is one of the motivations for a web-based tool.

Dynamic energy consumption causes the major voltage drop that appears on the switched capacitors. The slope of the continuous voltage drop corresponds to the leakage power consumption. The voltage across the two capacitors, C_{S1} and C_{S2} in Fig. 3, is

denoted by $V_{C1}(\cdot)$ and $V_{C2}(\cdot)$ respectively. The argument, $(\cdot)$, denotes the four different states of the capacitor supplying power (C_{S1}) to the target circuit. These are $(--)$, $(-)$, $(+)$ and $(++)$ which denote fully charged, connected to the on-chip bypass capacitor C_B, discharged by leakage energy consumption, and discharged by dynamic energy consumption. At the same time, C_{S2} is discharged during $(--)$ and remains in a fully charged state during $(-)$, $(+)$ and $(++)$ [10].

The static or leakage energy consumption is denoted by the slope of the waveform [11]. Let us denote the static energy of the i-th clock cycle by $E_s(i)$:

$$E_s(i) = \frac{1}{2}(C_{S1} + C_B)\frac{V_{C1}(i-)^2 - V_{C1}(i+)^2}{\Delta t}. \tag{1}$$

We eliminate Δt by converting the static power to energy consumption over the clock period τ. The dynamic energy of i-th clock cycle, $E_d(i)$, is denoted by

$$E_d(i) = \frac{1}{2}(C_{S1} + C_B)(V_{C1}(i+)^2 - V_{C1}(i++)^2). \tag{2}$$

Finally, the total energy consumption is given as

$$E = \sum_{i=0}^{n}(E_d(i) + \tau E_s(i)) = \sum_{i=0}^{n} E_d(i) + n\tau E_s. \tag{3}$$

It is not easy to determine the exact time that delimits the period for the dynamic energy, i.e., $V_{C1}(i++)$. Improper division into dynamic and static energy may cause severe errors if there are major changes in clock frequency. To avoid this, we measure the cycle-accurate energy at various clock frequencies. We cross-check the dynamic energy values measured at different clock frequencies and thus confirm the dynamic energy values [10].

6 Energy Measurement and Characterization of Memory Devices

As an example of memory device energy characterization, we introduce an SDRAM device, which are popular in everything from hand-held devices to desk-top computers. There are various operation methods, but we choose two major operations: burst-mode access and auto refresh (Fig. 5).

Table 2 shows Ξ of the SDRAM. Let $f_1(\cdot)$ be the number of 1s in $\cdot$. Among Ξ, the CD is the most dominant component. Table 3 shows the CD energy of K4S280832B-TC1L which are from a major vendor, Samsung. This has 128Mbit capacity with 4M address space $(12 \times 10) \times 8$bit data $\times$ 4 banks.

7 Energy Characterization of Off-Chip Buses

The energy consumption of synchronous off-chip buses is denoted by the energy state machines shown in Fig. 6. The driver specification is an LVT (Low-voltage TTL or BiCMOS, 3.3V) bus with 2.7pF transmission line capacitance. Low-voltage CMOS

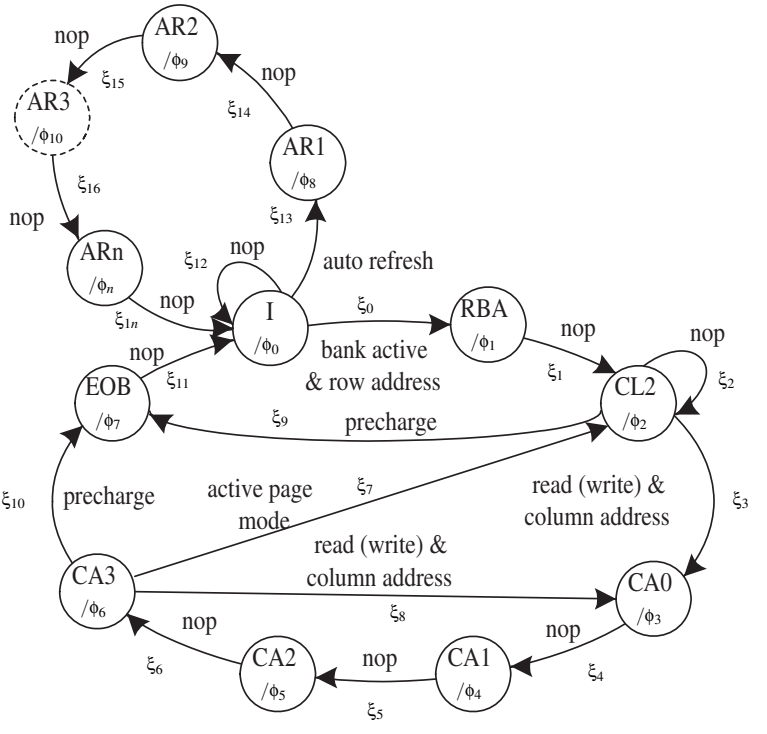


Fig. 5. Energy state machine of an SDRAM.

and other bus drivers such as SSTL (Stub-Series Termination Logic) will be provided in future [12]. To prevent the data bus from floating during a high-impedance state, bus-hold logic is selectable. Passive pull-up is not suitable for quality systems due to excessive static current, but we plan to include it anyhow. Bus-invert coding by the transition activities or by the logic-low state [12] can also be selected.

The set of states (s_0, s_1) in Fig. 6 (a) represents driven-low and driven-high states respectively. Fig. 6 (b) includes s_2 and s_3 to represent bus hold states. The power consumption of LVT and GTL+ bus and bus drivers have already been studied [12]. We compose an energy state machine for an LVT bus (these are commonly used high-performance memory buses for embedded systems) by converting the power values to cycle-accurate energy values. The bus hold logic acts like a small capacitance (typically 0.5pF) and consumes negligible DC current. We control the output enable no later than the input change in order to keep the rise and fall time constants. Thus no state change, such as $s_0 \rightarrow s_3 \rightarrow s_1$, is allowed. This guarantees that $e_4 = e_5 = e_6 = e_7 = 0$. In addition, $p_2 = p_3 = 0$.

When we composed a 2-inch bi-directional bus using a Fairchild 74LVT245, with $e_0 = e_1 = 0.55$, $e_2 = e_3 = 0$, $p_0 = 0.0053\tau$, and $p_1 = 0$ for Fig. 6 (a). The units are

Table 2. Dynamic energy consumption of SDRAM, Ξ (nJ/bit).

Ξ	Energy Cost
ξ_0,	$\Theta_{ra} + c_{ra} f_1(A_r)$
$\xi_1 = \xi_2 = \xi_7$	Θ_a
$\xi_3 = \xi_8$	read: $\Theta_{car} + c_{do} f_1(D_0) + c_{car} f_1(A_c)$ write: $\Theta_{caw} + c_{di} f_1(D_0) + c_{caw} f_1(A_c)$
$\xi_4 = \xi_5 = \xi_6$	read: $\Theta_{carb} + c_{do} f_1(D_i)$ write: $\Theta_{cawb} + c_{di} f_1(D_i)$
$\xi_9 = \xi_{10}$	Θ_{pr}
$\xi_{11} = \xi_{12}$	Θ_i
$\xi_{13} + ... + \xi_{1n}$	Θ_{rf}

Table 3. Common Mode Dynamic Energy, Θ (nJ/bit).

Symbol	Description	Value
Θ_{ra}	row active	1.500
Θ_{car}	column active (read)	0.800
Θ_{carb}	column active (read, burst)	0.186
Θ_{caw}	column active (write)	0.522
Θ_{cawb}	column active (write, burst)	0.033
Θ_{pr}	precharge	0.515
Θ_{rf}	refresh	4.941
Θ_a	active	0.021
Θ_i	idle	0.018

nJ/bit, and τ is the clock period of the synchronous bus in nS. These energy values are applied to address and data buses by using bus models described previously [13].

8 Energy Measurement Tool for FPGAs

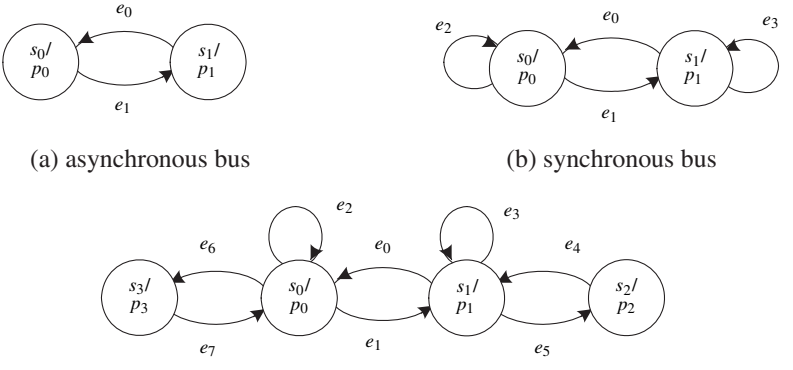
As the gate counts of FPGAs increase, they become more widely used for complex SoC design, of the final product as well as for rapid prototyping before taping out. As the power consumption of FPGAs can represent a significant proportion of that of the whole system, their power consumption behavior must be included in the primary characteristics that are to be taken into account in system-wide power reduction. Chip vendors are

Table 4. Coefficient of Ξ (nJ/bit).

Coefficient	Description	Value
c_{ra}	row address input	0.192
c_{do}	data output (read)	0.097
c_{di}	data input (write)	0.103
c_{car}	column address input to read	0.145
c_{caw}	column address input to write	0.161

Table 5. Static energy consumption, Φ (nJ/bit).

Φ		Value
$\phi_0 = \phi_8 = \phi_9 = \phi_{10} = \phi_n$		0.0016τ
ϕ_1		0.0051τ
ϕ_2		0.0049τ
ϕ_3	read	0.0017τ
	write	0.0087τ
$\phi_4 = \phi_5 = \phi_6$	read	0.0145τ
	write	0.0073τ
ϕ_7		0.0016τ



(a) asynchronous bus

(b) synchronous bus

(c) synchronous bus with bus hold

Fig. 6. Energy state machine of synchronous off-chip buses.

supposed to provide power consumption information for their products on the device data sheets. However, it is not possible for vendors to specify the exact power consumption of an FPGA design because of the dynamic power consumption. The device type, operating temperature and process variations largely determine the static power, which is fixed by the vendor during manufacture. In FPGAs, the main cause of static power dissipation is leakage. On the other hand, dynamic power consumption is completely design-dependent, and is determined by many factors including resource utilization and low-level features such as logic partition, mapping, placement and routing. The dynamic power consumption is also affected by the system-level behavior of the FPGA as it interacts with other devices, which is in turn determined by the microprocessor and application programs. High-fidelity power estimation must take all these factors into account.

The measurement circuit shown in Fig. 3 generates enormous data since it captures the energy values every 20ns. Thus computer controlled measurement system is mandatory for practical use of the measurement circuit. We have developed an in-house energy measurement tool for FPGAs. The tool is fully integrated with an automatic data acquisition system consisting of pipelined A/D converters, a vector generator, a host interface

through USB 2.0 communication channel and PC-based control software. Table 6 summarizes the specification of the in-house tool.

Table 6. Specification of the in-house measurement system.

Target FPGA: Xilinx Virtex-II XC2V1000FG456
and Spratan-II XC2S50TQ144
Target control FPGA: Xilinx Spartan-II XC2S150FG456
Data acquisition FPGA: Xilinx Spartan-II XC2S150FG456
Vector and configuration memory: Samsung SRAM 1MByte
Data acquisition memory: Samsung SRAM 1MByte
ADC resolution: 10 Bit ADC @50MS/s
Data transfer method: USB 2.0 communication

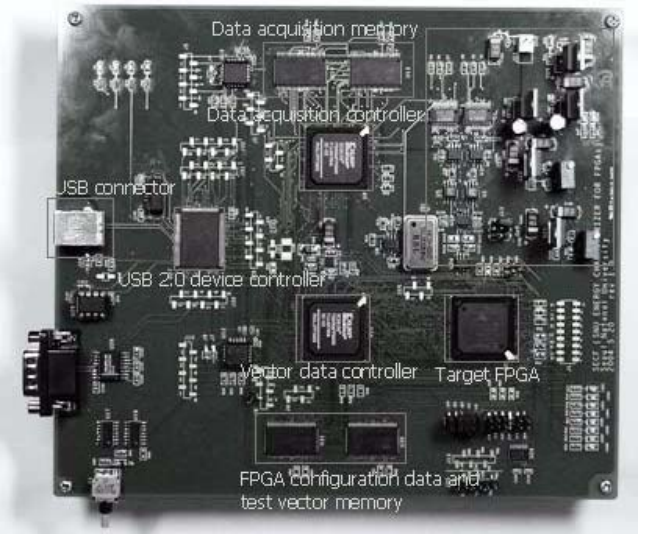


Fig. 7. SECF (SNU Energy Characterizer for FPGAs).

We have developed two different versions of tools. The first version is equipped with a Spartan-II device of small capacity (50K gates). Recently, we upgraded the target FPGA to a high-density Virtex-II device (1M gates). This requires replacement of the switch device to accommodate high power supply current. Fig. 7 shows a photograph of the tool. The tool has many convenient features such as bit-stream download without the Xilinx XChecker or JTAG cables, which simplifies the measurement process and enhances to ability to handle complex and repetitive measurements.

Table 7. Energy consumption for FIR filters (4 taps, dynamic: nJ/clock, static: mW, device: XC2V1000FG456).

Architecture		XPower			Multimeter			Our tool		
		Dynamic		Static	Dynamic		Static	Dynamic		Static
		Ave.	Max.		Ave.	Max.		Ave.	Max.	
TCS	Direct form	8.61	NA	150	2.39	NA	14.44	2.88	3.73	14.63
	Transposed form	5.47	NA	150	1.63	NA	14.59	1.67	2.83	14.82
RNS	Direct form	14.53	NA	150	4.40	NA	15.32	4.29	5.67	15.55
	Transposed form	11.58	NA	150	3.91	NA	15.46	3.94	5.08	15.50

Digital filters such as the IIR and FIR filters are widely used in signal processing. Various architectures have been proposed to achieve high performance, small size and low power. We measured and compared the energy consumption for FIR filters with different architectures. Table 7 shows energy consumption of 4 types of FIR filters. We targeted TCS (Two's Complement System) as a traditional architecture and RNS (Residue Number System) as an advanced architecture. Generally, an RNS architecture is known to consume less power when the number of taps is greater than n, due to the extra logic needed for binary to RNS converters and vice versa. Since the energy consumption is linearly proportional to the number of taps, we could estimate the energy consumption of an FIR filter with larger n through repeated experiments, and on this basis the minimum number of taps turned out to be 18. We were also able to confirm that transposed forms are superior to directed forms, both for TCS and RNS architectures. As described in the previous experiment, XPower produces higher estimates, for similar reasons.

9 Energy Measurement Tool for a RISC Processor

The Seoul National University energy scanner (SES) is a highly integrated, energy monitoring tool for ARM7TDMI RISC processors that collects power consumption data in a cycle-by-cycle resolution and associates the collected power data with C program and assembly language source code. SES does not require any additional measurement equipment because the power measurement circuitry is embedded in its board. By presenting energy-monitoring results at the C source or assembly language levels using the GNU project debugger (GDB)-like user interface, SES helps users identify potential energy hot spots in embedded programs. The current version of SES works for ARM7TDMI-processor-based embedded systems. However, the proposed power measurement technique and its overall energy-monitoring methodology are both platform independent.

SES has three logical modules: energy estimation, energy analysis, and user interface. The energy estimation module consists of the energy measurement board and the memory energy estimator. The board is a peripheral component interconnect (PCI) bus expansion card that uses a real-time profile acquisition module to collect a target application's cycle-accurate system traces. The PCI local-bus interface transfers the collected system traces to the host PC, which runs a Linux operating system. The en-

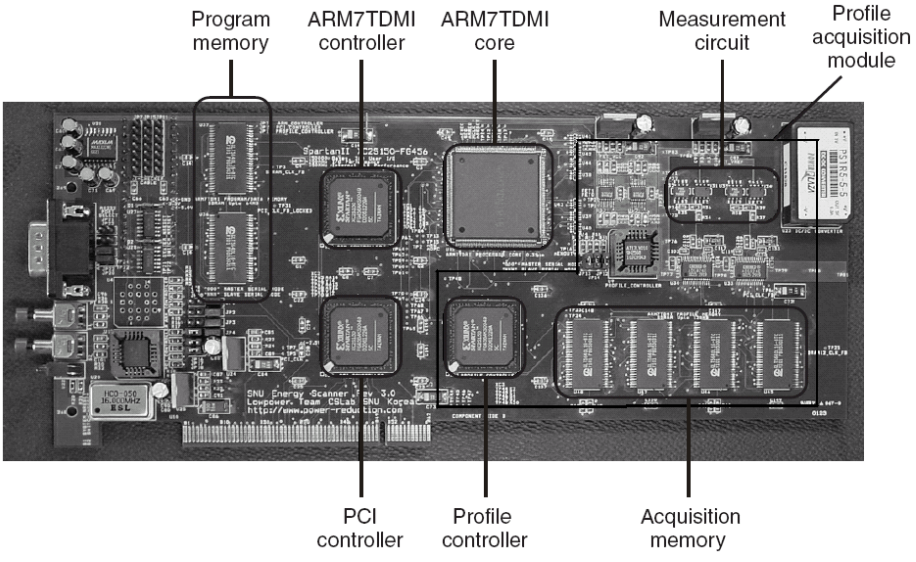


Fig. 8. SES (SNU Energy Scanner).

ergy measurement board includes the ARM7TDMI processor core with its controller, profile acquisition module, program memory, and PCI controller, as Fig. 8 shows. The profile acquisition module consists of the cycle-accurate energy measurement circuit, acquisition memory, and profile controller. The energy measurement board works as an ARM7TDMI emulator equipped with the cycle-accurate energy measurement circuit. A system trace collected from the board includes a cycle-level energy trace of the processor core and a cycle-level memory trace. The memory energy estimator running on the host PC is a software memory simulator with cycle-accurate energy models for various caches, memory buses, and memory devices. The measurement board transfers the memory traces as inputs to the memory energy estimator and the estimator produces the cycle-level energy profile of the off-chip memory system and cache memory. The energy analysis module matches the cycle-level energy profile of the target processor and memory system to the program's source code. The module associates the energy profile with the source code at three different levels: C source, assembly language, and C function.

10 Integrated Energy Estimation Tool

SNU Energy Explorer (SEE) is an in-house system-level energy exploration tool that executes real application software on a testbench based on SES [14] With the increasing trend towards low-power design, a higher-fidelity, system-level energy estimation environment is demanded. In this paper, we propose a web-based energy exploration tool, SEE Web [15], which overcomes limitations of existing tools. For cycle-accurate energy

estimation, a FSM (Finite-State Machine) model that isolates dynamic and leakage energy consumption has been developed with the aid of a cycle-accurate measurement technique [13], [11]. Additionally, the ISS (Instruction Set Simulator) is built in hardware to increase the simulation speed, and the tool is now available to the public on the Web. To achieve a wider design space, SEE Web leaves many things available to be configured by users. Some of the configurable parameters are the processor clock frequency, cache organization, and the SDRAM control policy. Although SEE is a useful tool, it has serious limitations for open use because it includes custom hardware. SEE Web has overcome this limitation through web technology, providing a high-fidelity energy estimation environment to any Internet user.

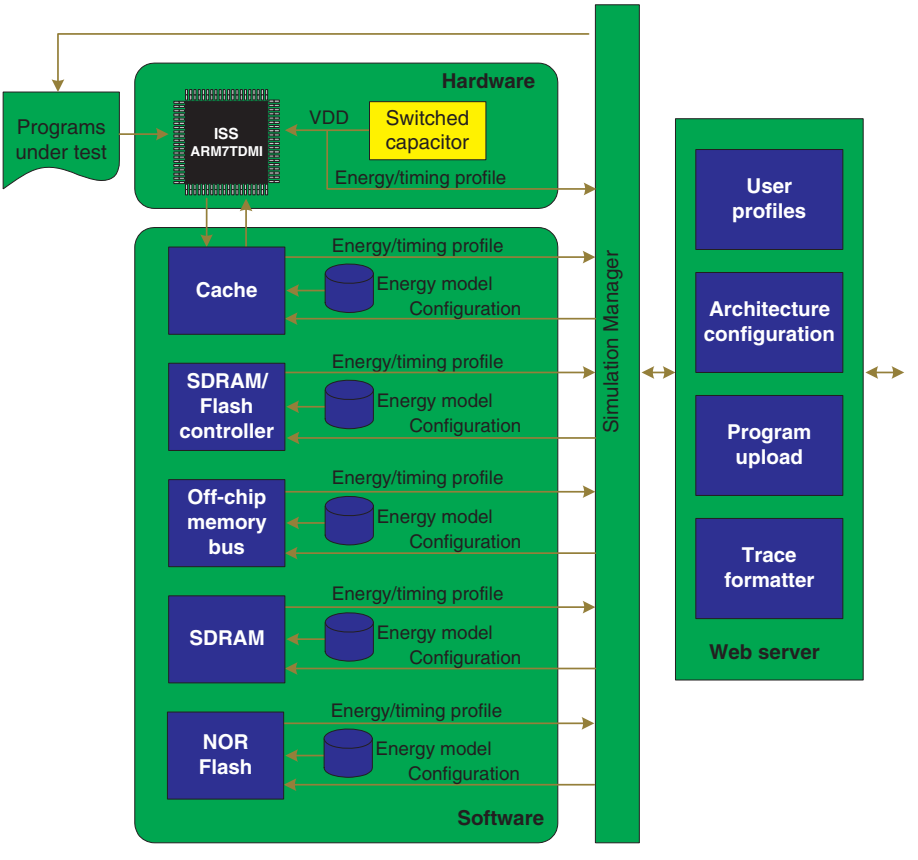


Fig. 9. SEE Web (SNU Energy Explorer Web).

SEE Web is expandable, and a new IP can be added in the form of a behavioral-level C code function. Such a function will include an energy consumption model at the clock-cycle level, in the form of an energy FSM; this is a distinct feature of our tool and

enables design space exploration with complete freedom. Fig. 9 shows the architecture of SEE Web. A typical embedded system composed of a CPU, cache, off-chip bus, SDRAM controller and SDRAM is connected to a web server through the simulation manager. Users can configure the system architecture, upload a program to simulate, and see the simulation result through the web interface.

11 Conclusions

We have presented a high-fidelity energy exploration tool aimed at over-layer energy optimization of embedded systems. System components are modeled as finite-state machines, associating transitions with dynamic energy and states with leakage power. The superior modeling ability of the energy state machine enables precise energy estimation while providing a fast and user-friendly environment for system designers who are not familiar with device technologies. The series of energy measurement and estimation tools are easy and free energy exploration environment which encourages users without detailed knowledge to perform a system-level energy optimization. Among them, SEE Web is the publicly promoted version of our accurate in-house energy estimation tool SEE and is now available on the web at http://see.snu.ac.kr. Our tool development is an active project and will be regularly maintained and upgraded.

References

1. C. X. Huang, B. Zhang, A.-C. Deng, and B. Swirski, "The design and implementation of powermill," in *Proceedings of International Workshop on Low Power Design*, pp. 105–110, Apr. 1995.
2. J. Flinn and M. Satyanarayanan, "Powerscope: a tool for profiling the energy usage of mobile applications," in *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 2–10, Feb. 1999.
3. W. R. Hamburgen, D. A. Wallach, M. A. Viredaz, L. S. Brakmo, C. A. Waldspurger, J. F. Bartlett, T. Mann, and K. I. Farkas, "Itsy: Stretching the bounds of mobile computing," *IEEE Computer*, vol. 34, pp. 28–37, Apr. 2001.
4. V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Tranactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, pp. 437–445, Dec. 1994.
5. A. Sinha and A. Chandrakasan, "Jouletrack - a web based tool for software energy profiling," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 220–225, June 2001.
6. W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 340–345, June 2000.
7. D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proceedings of International Symposium on Computer Architecture*, pp. 83–94, June 2000.
8. T. Simunic, L. Benini, and G. de Micheli, "Energy-efficient design of battery-powered embedded systems," *IEEE Tranactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, pp. 15–28, Feb. 2001.
9. N. Chang, K.-H. Kim, and H. G. Lee, "Cycle-accurate energy measurement and characterization with a case study of the ARM7TDMI," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, pp. 146–154, Apr. 2000.

10. H. G. Lee, S. Nam, and N. Chang, "Cycle-accurate energy measurement and high-level energy characterization of FPGAs," in *Proceedings of 4th International Symposium on Quality Electronic Design (ISQED 2003)*, pp. 267–272, Mar. 2003.
11. A. Sinha and A. Chandrakasan, "Energy aware software," in *Proceedings of the 13th International Conference on VLSI Design*, pp. 50–55, Jan. 2000.
12. N. Chang, K.-H. Kim, J. Cho, and H. Shin, "Bus encoding for low-power high-performance memory systems," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 800–805, June 2000.
13. H. Shim, Y. Joo, Y. Choi, H. G. Lee, and N. Chang, "Low-energy off-chip sdram memory systems for embedded applications," *ACM Transactions on Embedded Computing Systems*, vol. 2, pp. 98–130, Feb. 2003.
14. D. Shin, H. Shim, Y. Joo, H.-S. Yun, J. Kim, and N. Chang, "Energy monitoring tool for low-power embedded programs," *IEEE Design and Test of Computers*, vol. 19, pp. 7–17, July-Aug. 2002.
15. I. Lee, Y. Choi, Y. Cho, Y. Joo, H. Lim, H. G. Lee, H. Shim, and N. Chang, "Web-based energy exploration tool for embedded systems," *IEEE Design and Test of Computers*, vol. 21, pp. 572 – 586, November – December 2004.

CODACS Project: A Development Tool for Embedded System Prototyping

Lorenzo Verdoscia

Institute for High Performance Computing and Networking (ICAR) - CNR
Via Castellino, 111 - 80131 Napoli, Italy
lorenzo.verdoscia@na.icar.cnr.it

Abstract. The advent of FPGAs and Intellectual Property core availability allow great freedom in the customization of platform processors for embedded systems. One of the new challenges that such technologies present is how to implement a high performance application on devices with hundreds coarse-grained computing units running at 200 MHz, rather than on one processor running at 20 GHz. Consequently, to profit by spatial parallelism that such devices offer becomes a non marginal issue. From the architectural point of view, at least two questions arise: how to exploit such spatial parallelism; how to program such platforms. The first one brings us to seriously reconsider the dataflow paradigm, given the fine grain nature of its operations. The second one brings us to seriously reconsider the functional programming style, given its inherent simplicity in writing parallel programs. In this paper we will discuss our experience in combining these two approaches inside CODACS (COnfigurable DAtaflow Computing System) demonstrator. The resulting architecture offers interesting properties not only as stand-alone computing system but also as development tool for Application Specific Processor (ASPs) prototyping activities.

Key words: FPGA, dataflow computing, functional programming, Application Specific Processor (ASP), embedded system.

1 Introduction

In several real-time applications, custom processors based on application-specific or domain-specific instruction sets are gaining popularity to speed up the application and are often used to implement critical architectural blocks in complex system-on-chips (SoC). Improvements in semiconductor fabrication technologies promise to make it feasible to replace logic gates or hardware macro-blocks with microprocessors as building blocks for integrated circuit (IC) design. Such programmable solutions will provide the ability to meet short product cycles and cope with changing application functionality (e.g., in areas with evolving standards). However, the rapid expansion in the market for embedded systems with tight constraints on cost, performance, size, and power consumption implies that the need to customize the architecture to the application or application domain will continue to be a primary driving requirement in system-on-chip design.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 59–64, 2005.

While several advances have been made in custom processor architectures [6,3,8], tools [13], and design methodologies [1], designers are still required to manually perform some critical tasks, such as selection of the custom instructions best suited to the given application and design constraints. This is a highly fallible undertaking and may require formidable resources.
Even if development is completed on time, considerable consistency problems may appear between hardware implementation and software development tools. Furthermore, a small change in design requirements or a bug during later stages can ensue in daunting amounts of labor. Consequently, the prototyping phase is crucial in the development of such processors, and adequate hardware/software development tools become often requires.

In this paper, we present our experience with CODACS demonstrator [10] as prototyping tool for custom processor design. It combines the functional programming paradigm [11] and the dataflow execution model [12] for reconfigurable computing that is decoupled from details of the underlying platform. Application-level programmers should be liberated from reconfigurable hardware accelerator details. Dually, hardware designers should not be exposed to interfacing details. This is the way we believe reconfigurable computing must take in order to become mainstream.

2 CODACS Demonstrator

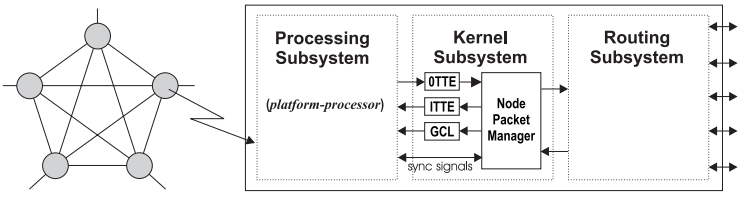


Fig. 1. CODACS Architecture connected as WK-recursive with $N_d = 5$ and $Level = 1$.

CODACS (COnfigurable DAtaflow Computing System) project goal is to build up a high performance reconfigurable computing system demonstrator able to efficiently execute dataflow processes obtained compiling programs written in CHIARA language [11] (a Backus'FP [5] functional programming language dialect) and oriented to make easier the Application Specific Processor design. Main features of CODACS are: highly scalable architecture, reconfigurable dataflow computing environment (*platform-processor*), and functional assembly language to program it. Figure 1 shows CODACS architecture. The prototype is constituted by a Gidel PROC20KE board [7] with 5 Altera APEX20K15-3C [4] FPGA components. Each node has been partitioned into three concurrently operating subsystems:

- Processing Subsystem (called *Platform-Processor*), devoted to execute the dataflow graph assigned to the node on the basis of information received from the Kernel Subsystem;
- Kernel Subsystem, devoted to unpack a message, manage processor configurations and related I/O data tokens, and prepare new messages for the Routing Subsystem;
- Routing Subsystem, devoted to provide all routing functions for incoming and outgoing messages. When a message reaches this module, its header is processed to be routed towards either the appropriate output link or the Kernel Subsystem.

2.1 Platform-Processor

The platform-processor is the reconfigurable core of CODACS prototype, and it has been tailored to execute in hardware dataflow graphs obtained compiling CHIARA programs. Its design process, schematized in Figure 2, has been based on the language first approach in conjunction with the *homogeneous* High-Level Dataflow System (*h*HLDS) model [12]. As result, we implemented a platform-

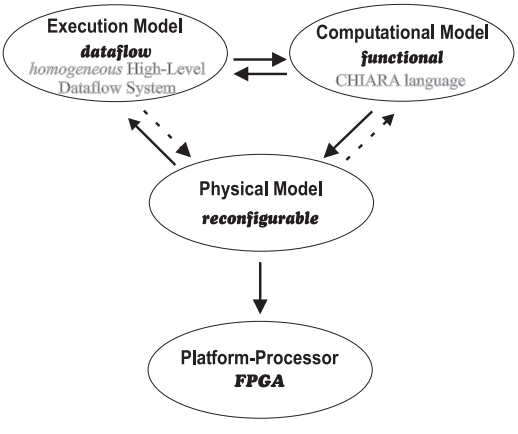


Fig. 2. Platform-Processor design methodology.

processor (Figure 3) composed of: a) 64 identical MultiPurpose Functional Units (MPFU); b) the MPFU Interconnect that allows to connect any MPFU output to any other MPFU input and each register of the Token Ensemble Buffers to the corresponding MPFU; c) the Control; d) three banks of I/O buffers for token transfer respectively named TOKEN_IN A and B (to store, for each MPFU, respectively the right and left token coming from the Kernel Subsystem) and TOKEN_OUT (to transfer results to the Kernel Subsystem); e) the Graph Setter that stores the Graph Configuration table ready to be executed.

As token loading can be overlapped to computation operations, a platform-processor can, when different tokens are applied to the same configuration, execute pipeline activities simply checking data flow from/to the Kernel Subsystem. MPFUs only compute binary operations belonging to the functionally complete set of elementary CHIARA operators. Such operators include commonly used arithmetic and logic operators and the new operators **LST** (loop start), **SL** (select left), and **SR** (select right). Furthermore, it executes dataflow graphs a) without using memory to store intermediate tokens when they flow from a Computing Unit to another, reducing thus continual LOAD and STORE operation and memory latency problem; b) in a completely asynchronous manner, disposing of a straightforward data flow control and an operation firing mechanism at a minimal hardware cost. We have augmented the token with the concept of *validity*, coded by one bit, that denotes whether data is ready to be processed or not.

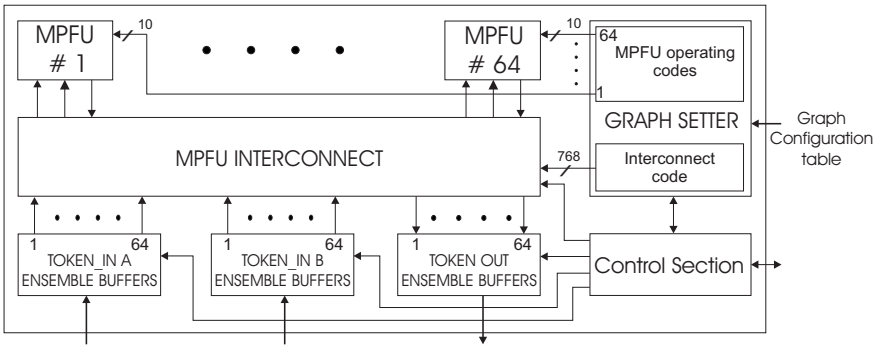


Fig. 3. Platform-Processor block scheme.

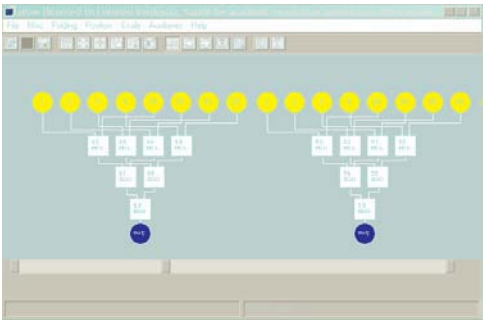
2.2 CHIARA Compiler

CHIARA language peculiarity is that it defines a set of elemental operators that is *functionally complete*, i.e. able to generate any other more complex function of the language. This set constitutes the assembly language for the platform-processor and is directly implementable in hardware. After compiled, a CHIARA program is translated in the *Dataflow Graph Description* (DGD) table that represents the program dataflow graph. This table details, for each node of the graph, the following information: the number assigned; the operation code; a tag to identify if the right and left input data is an initial value or the output data of another node, reporting in this case the number of the sender node; a tag to identify if the output data is a final or intermediate value. In the second case, the number of the receiver node is reported. Figure 4.a and b. respectively shows a piece of a DGD table produced by the compiler for a matrix multiplication

$A(4,4) \times B(4,4)$ and the corresponding dataflow produced by aiSee [2] a software tool based on the graph layout tool VCG (Visualization of Compiler Graphs) [9] and designed to explore huge graphs (containing hundreds, thousands, and sometimes even hundreds of thousands of elements).

Node#	Func	Apply level	Constr level	Insert Level	Left In	Right In	Out
..	...	.	.	.	..	..	..
43	MUL	1	0	0	%1	%30	47
44	MUL	1	0	0	%2	%30	47
45	MUL	1	0	0	%3	%30	48
46	MUL	1	0	0	%4	%30	48
47	ADD	0	0	1	43	44	49
48	ADD	0	0	1	45	46	49
49	ADD	0	0	2	47	48	out
50	MUL	1	0	0	%1	%40	54
51	MUL	1	0	0	%2	%40	54
52	MUL	1	0	0	%3	%40	55
53	MUL	1	0	0	%4	%40	55
54	ADD	0	0	1	50	51	56
55	ADD	0	0	1	52	53	56
56	ADD	0	0	2	54	55	out
..	...	.	.	.	..	..	..

a)



b)

Fig. 4. Part of compiler outcome for a matrix multiplication $A(4,4) \times B(4,4)$: a) DGD table; b) iSee visualization.

2.3 Program Running

A program execution takes place when the related dataflow graph is loaded onto a platform-processor. To do this, the program dataflow graph is partitioned into subgraphs wholly mappable onto a platform-processor, each subgraph is first transformed into a Graph Configuration and Input Token Value table and then stored in the GCL and ITTE of the Kernel Subsystem. Afterwards, these tables are ready to be loaded into the platform-processor on the basis of the scheduling policies. Since the Graph Configuration and Input Token tables of a subgraph split the connection between data and related operations, subgraph run can take place but the loading of the two tables can occur at a different time. An immediate benefit is that we can overlap configuration loading activities and processor execution.

3 Conclusions

Custom processor development is quite challenging for real-life applications because of tradeoffs involving during its design. The system described within this paper represents an ambitious set of goals for a design tool. Our approach is based on the usage of CHIARA functional language to describe the application that the custom processor would execute, the *homogeneous* High Level Dataflow System model to define the rules to obtain the application dataflow graph, and finally CODACS platform-processor to validate the custom processor according to application specifications.

References

1. A. Abbas, S. Khan, and M. Usman. Optimal application specific processor and development tool design methodology. In *Proc. IEEE Intl. Multi Topic Conference (INMIC)*, Karachi, Pakistan, dec 2002. IEEE Press.
2. AbsInt. aisee. www.AbsInt.com/aisee.
3. O.T. Albahama, P. Cheung, and T.J. Clarke. On the viability of FPGA-based integrated coprocessors. In Pocek K.L. and Arnold J., editors, *Proc. IEEE Symp. FPGAs for Custom Computing Machines*, pages 206–215, April 1996.
4. ALTERA Corporation. APEX 20K devices: System on a programmable chip solutions. http://www.altera.com/products/devices/apex/apx-index.html, 2001.
5. J.W. Backus. Reduction languges and variable free programming. Technical Report RJ-1010, IBM, Yorktown Heights, NY, April 1972.
6. K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 2002.
7. GIDEL LTD. PROC20KE board. www.gidel.com, May 1999.
8. S.D. Haynes, J. Stone, P.Y.K. Cheung, and L. Wayne. Video image processing with the sonic architecture. *Computer*, 33(4):50–57, April 2000.
9. G. Sander. VCG visualization of compiler graphs. Technical Report A01-95, Universität des Saarlandes, FB 14 Informatik, 66041 Saarbrücken, Germany, February 1995.
10. L. Verdoscia. CODACS project: A demand-data driven reconfigurable architecture. In *Euro-Par 2002*, volume 2400 of *LNCS*, pages 547–550, Paderborn, Germany, August 27–30, 2002. Euro-Par Conference Series, Springer-Verlag.
11. L. Verdoscia, M. Danelutto, and R. Esposito. CODACS prototype: CHIARA language and its compiler. In *Proceedings of the First International Workshop on Embedded Computing*, Tokyo University of Technology, Hachioji, Tokyo, Japan, March 23–26, 2004. IEEE Computer Society Press.
12. L. Verdoscia and R. Vaccaro. A high-level dataflow system. *Computing*, 60(4):285–305, 1998.
13. M. Vuletic, L. Pozzi, and P. Ienne. Development Environment for Dynamically Reconfigurable Embedded Systems. In *15th IEEE Intl. Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)*, pages 339–351, Galveston, Texas, September 27–29, 2004. IEEE Press.

A Study on Web Services Selection Method Based on the Negotiation Through Quality Broker: A MAUT-based Approach

Young-Jun Seo[1], Hwa-Young Jeong[2], and Young-Jae Song[1]

[1] Department of Computer Science, Kyung Hee University
1 Seocheon, Giheung, Yongin, Gyeonggi 449-701 KOREA
{yjseo, yjsong}@.khu.ac.kr
[2] Department of Multimedia Design, Yewon Art University
271 Changinree Shinpyungmyun Eymshilgun JonrabukDo, KOREA
jhymichael@empal.com

Abstract. In web service area, which is growing fast recently, because service discovery is restricted only by functional requirement, the optimal web service selection method considering quality, a non-functional element, is regarded as important. In this research, we suggested Web Service Quality Broker Architecture including Quality Broker, which provides quality negotiation environment, and proposed web service selection method, which helps service requester find the service provider which gives the maximum benefit and bind that service dynamically. We described the internal negotiation procedure for quality attributes of user's point of view with Multi-Attribute Utility Theory (MAUT) and linear programming.

1 Introduction

Recently the web service area providing lots of advantages like platform independency, interoperability, easy service usability and so forth is rapidly burgeoning as the next generation IT paradigm. Ovum estimates that the average growth of the web service market will be at 118% per year and that the market, in 2006. It will consist of 70% of Professional Service, web service consulting/construction area, and 18% of Hosted Service, web service transfer area [1].

Web service consists of XML-based platform, component based distributed computing technology independent of implementation language and three kinds of roles performing publish-find-bind operations. Each role is performed by the web service provider providing web service, the web service requester using web service and UDDI registry helping service requester search the detailed specifications of public web services. In the existing web service model, the role of UDDI registry is restricted as service finding for only functional requirement and there is a defect that 48% of UDDI registry has a connection including lost, broken or incorrect information [2]. Under these circumstances, optimal web service selection is a complicated problem to requester and lots of quality attributes must

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 65–73, 2005.

be measured and evaluated simultaneously. Since many quality attributes are related with each other, the improvement of one quality attribute may cause the deterioration of the other quality attribute [3]. Patrick [4] presented the optimal common gains between provider and requester using logrolling, negotiation strategy about QoS and CoS attribute. But, in the negotiation stage, the only quantity of QoS attribute was measured without consideration of sub-attributes included in QoS attribute.

In this paper, we explain the concept about Web Service Quality Broker Architecture with Quality Broker role, which provides the existing web service architecture with negotiation environment, and proposes the web service selection method, which, by calculating the maximum gains on the basis of the utility function of mutual quality attributes between service requester and service provider, finds the service provider with the optimal quality and helps bind dynamically.

2 Related Works

2.1 WSLA (Web Service Level Agreement)

WSLA developed by IBM in 2001 is the framework to define and monitor obligations to service providers and requesters [5,6]. It measures and monitors the QoS parameters, checks the agreed-upon service levels, and reports violations to the authorized parties involved in the SLA management process.

It comprises the Parties, Service Definitions, and Obligations sections. The parties section, consisting of the signatory parties and supporting parties fields, identifies all the contractual parties. The service definitions section specifies the characteristics of the service and its observable parameters as follows. Examples of such SLA Parameters are "availability", "throughput", and "response time". Metrics describes the formula for the calculation about quality factors stated, SLA Parameters are composed of (composite) Metrics, which, in turn, aggregates one or more other (composite) metrics, according to a measurement directive or a function. Obligations, consisting of the Validity Period, Predicate, Actions, define various guarantees and constraints that may be imposed on the SLA parameters.

2.2 Negotiation Strategy

Negotiation is a decision process in which two or more parties make individual decisions and interact with each other for mutual gain [7]. Negotiation includes a set of tasks, such as problem definition, generation of alternatives, evaluation of alternatives, and preference modeling, that are executed by a set of parties. In particular, each negotiation involves at least two parties. On the other hand, negotiation involves a set of issues and every issue contains a set of alternatives. Furthermore, the set of issues may also be constrained by a set of criteria.

One of the strategies for achieving negotiation is called MAUT. MAUT (Multi-Attribute Utility Theory) concretes utility function and attribute reflecting individual stand for risks to the model and then identification for related appropriate function's form decides individual preference and utility function's form [8]. MAUT evaluates the gains, considering the multi-attribute for the subject of negotiation, and each attribute has a relative weight to other attributes. In MAUT, utility function is represented using weight (w_j) and evaluation function ($v_j(x[j_i])$) for each attribute. Utility function representing the gains of service requester and provider can be described as equation (1) and (2).

$$RequesterUtility = \sum_{i=1}^{n} w_j^r v_j^r(x[j_i]) \quad 0 \leq RequesterUtility \leq 1 \ . \tag{1}$$

$$ProviderUtility = \sum_{i=1}^{n} w_j^p v_j^p(x[j_i]) \quad 0 \leq ProviderUtility \leq 1 \ . \tag{2}$$

In order to standardize the satisfaction of suggested value about each attribute within the value from 0 to 1, we propose the evaluation function having request value and allowable value as the boundary like equation (3) and (4).

$$v_j^k(x[j_i]) = \frac{x[j_i] - allowable}{request - allowable} \quad \text{if} \ \ request \geq allowable\ value \ . \tag{3}$$

$$v_j^k(x[j_i]) = \frac{allowable - x[j_i]}{allowable - request} \quad \text{if} \ \ request < allowable\ value, (k = r, p) \ . \tag{4}$$

From equation (1) through (4), we set up the function maximizing the gains of service requester and provider as the object function and the condition that a difference between request and provider is in a certain aberration as a constraint, and presented linear programming like equation (5) through (7). In this paper, we refer to related work [9] in electronic commerce domain in order to set up the linear programming model to obtain the optimal value and the utility function for the suggested value between service requester and provider.

$$Object\ function : Max\ Z = RequesterUtility + ProviderUtility \ . \tag{5}$$

$$Constraint : |RequesterUtility - ProviderUtility| \leq threshold(= 0.01) \ . \tag{6}$$

$$Boundary : min.ofcommonrange \leq x[j_i] \leq max.ofcommonrange(i = 1, 2, 3) \ . \tag{7}$$

3 Web Services Selection Method

3.1 Web Service Quality Broker Architecture

Web Service Quality Broker Architecture proposed in this paper has different characteristics from the architecture proposed in the related work [10]. Currently

there are two classes of Service Providers providing web services [11]. Services from the first class are built with QoS support, referred to as QoS servers. They have the ability to assign different amount of system resources to different clients according to their QoS requirements. The QoS information supplied by QoS servers includes service levels with corresponding costs, maximum service capacities and currently available capacities at each service level. The other class of Service Providers are not built with QoS support and called legacy servers. There is no service quality level concept in legacy servers. In this paper, we considered the Service Provider as the QoS server that supplies QoS service levels. Service Requester can obtain the service providing the quality with requested level from WSLA of Quality Broker created by pre-monitoring. Rough quality warranty process including WSLA Service [6] in Web Service Quality Broker Architecture is like follows.

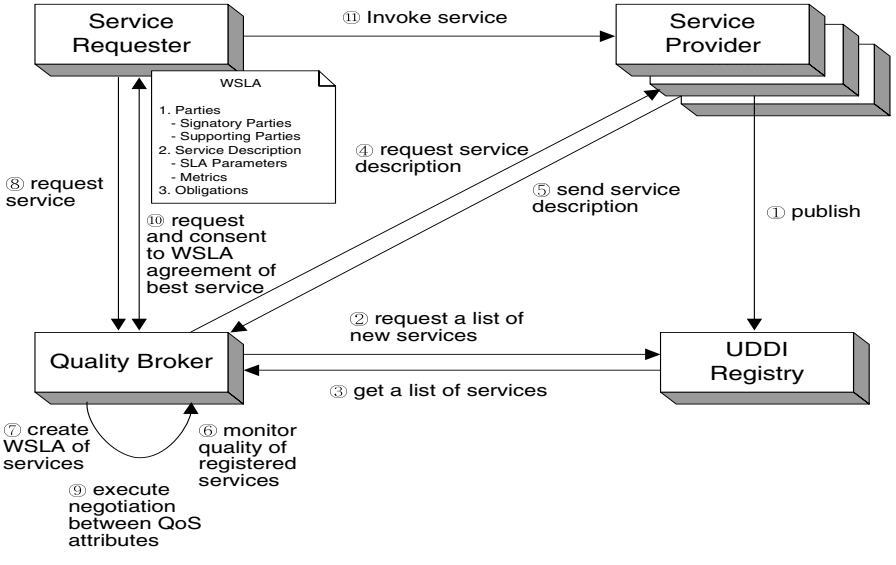


Fig.1. Interactions between the four participating roles

A list of Service Providers registered newly in UDDI registry is sent to Quality Broker, and Quality Broker requests Service Provider's service descriptions (WSDL files) in the list. Quality Broker monitors quality attribute values of registered service on the basis of the transferred service descriptions. The calculating result from monitoring and WSLA Metrics is stored in WLSA document, and this step is repeated periodically. Service Requester requests the service to Quality Broker with quality attribute value, and Quality Broker performs the negotiation through investigating WSLA details of services with same function and quality attributes of requester. When, from the result of negotiation, best

service is decided, the fact is transmitted to Service Requester, and contract is completed, if Requester agrees with WSLA of corresponding service. The outcome of the negotiation process is a single SLA document comprising the relationship and obligations of all the involved signatory parties. The WSLA Service distributes the SLA document available for deployment to the involved parties. In last, Service Requester requests the service to selected Service Provider. The WSLA Service measures SLA parameters such as availability or response time either from inside or outside the service provider's domain. It obtains measured values of SLA parameters and compares them against the thresholds given in the SLA. This can be done periodically. Once the result of comparison has been violated, the WSLA Service will carry out the appropriate actions to correct the problem as specified in the SLA.

3.2 Web Service Quality Model

In most of web service quality model [12,13], since the quality of requester's point of view in use about web service acts as the important decision factor due to web service characteristics. The quality in use among software qualities defined

Table.1. Web Service Quality Specification

Classification		Description	Service Level		
			gold	silver	bronze
Performance	Response Time(ms)	The average time required to complete a service request	0.1 ≤ RT ≤ 0.3	0.4 ≤ RT ≤ 0.7	0.8 ≤ RT ≤ 0.9
	Throughput (requests/s)	The number of completed service requestors over a time period	151 ≤ Th ≤ 200	101 ≤ Th ≤ 150	51 ≤ Th ≤ 100
Safety	Availability (probability)	quality aspect of whether the Web service is present or Ready for immediate use	0 ≤ Av ≤ 0.3	0.4 ≤ Av ≤ 0.6	0.7 ≤ Av ≤ 1
	Reliability (probability)	The ability of a service to perform its required Functions under stated conditions for a specified period of time	0.7 ≤ Re ≤ 1	0.4 ≤ Re ≤ 0.6	0 ≤ Re ≤ 0.3
Cost	Cost(€)	Cost involved in requesting the service	0.05	0.03	0.01

by ISO 9126 is mostly dealt with. Web service quality in use is divided into performance quality aspects, safety quality aspects, middleware service quality aspects, manageability quality aspects, and interoperability quality aspects. However, in this paper, we consider only performance and safety quality aspects among them and cost.

<Table 1> describes quality attributes and descriptions included in each web service quality aspects, and, also, presents various service levels[13] which service providers offer. The value of service level of availability and reliability belonging to safety quality aspects was described with the value of ratio of usable time and average time when failure occurs in total monitoring time according to three level ranges.

3.3 Negotiation Model and Procedure

In this paper, we consider multi-party and multi-issue negotiation, which contains service requester, several service providers and attributes. We focus on the method, which finds the optimal compromise suggestion between both sides and decides the service provider offering the maximum gains.

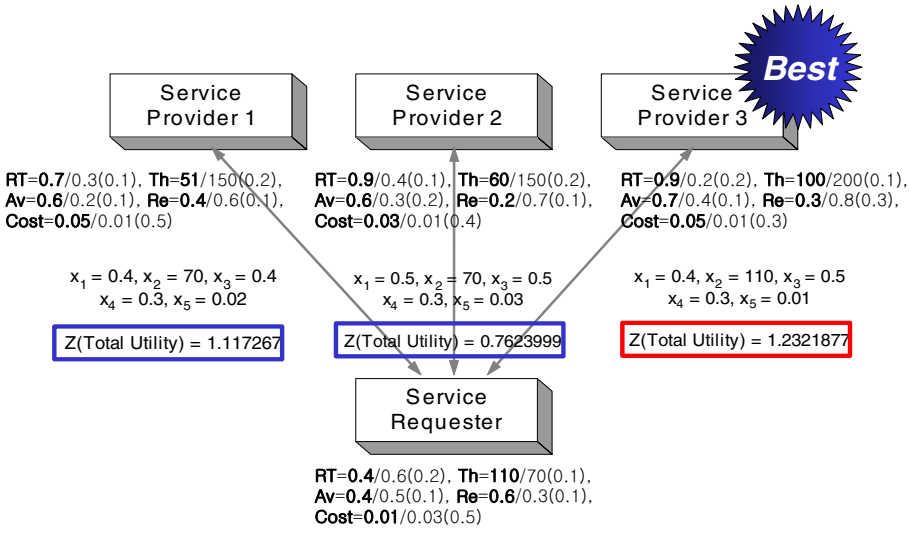


Fig.2. Result of Web Services Selection

Here, we define the optimal compromise suggestion as the proposal that warrants the maximum common gains to each other. In Negotiation model, we assume that service providers have different attributes for web service providing same function and service requester is a consumer who purchases web service with specific function offered by service providers. As attributes for Negotiation, response time, throughput, availability, reliability and cost are considered. Service requester and service provider have the weight (values in the parentheses), which is the relative preference value of specific attribute to other attributes. Generally, qualities offered by service provider are divided into three service levels (cf. Table 1), and requester has no choice but to select only one level

according to proposed cost. This paper includes the desired request value for specific quality attribute and the allowable value which can be permitted to the opposite side through negotiation between both sides. Service requester has low response time, high throughput, low availability, high reliability and low cost as request values, and provider is set up in the reverse way.

Fig. 2 shows attributes, which are possessed by one service requester and three service providers and the negotiation results. The value used in Fig. 2 as request (bold strokes) and allowable value (normal strokes) is represented by composing values in each service level provided differentially per each quality attribute. Supposing variable x_1 to x_5 are response time, throughput, availability, reliability, and cost, respectively. The gains of Requester and Provider1 in Fig. 2 can be represented by utility function[9] as follows.

$$RequesterUtility = \frac{0.6 - x_1}{0.6 - 0.4} \times 0.2 + \frac{x_2 - 70}{110 - 70} \times 0.1 + \frac{0.5 - x_3}{0.5 - 0.4} \times 0.1 \quad (8)$$

$$+\frac{x_4 - 0.3}{0.6 - 0.3} \times 0.1 + \frac{0.03 - x_5}{0.03 - 0.01} \times 0.5$$

$$Provider1Utility = \frac{x_1 - 0.3}{0.7 - 0.3} \times 0.1 + \frac{150 - x_2}{150 - 51} \times 0.2 + \frac{x_3 - 0.2}{0.6 - 0.2} \times 0.1 \quad (9)$$

$$+\frac{0.6 - x_4}{0.6 - 0.4} \times 0.1 + \frac{x_5 - 0.01}{0.05 - 0.01} \times 0.5$$

The above problem can be transformed into linear programming form [9] as follows.
Object function:

$$Max\ Z = -0.75x_1 - 0.0005x_2 - 0.75x_3 + 0.167x_4 - 12.5x_5 + 2.028 \quad (10)$$

Constraint:

$$-1.25x_1 + 0.0045x_2 - 1.25x_3 + 0.833x_4 - 37.5x_5 \leq -1.112 \quad (11)$$

$$-1.25x_1 + 0.0045x_2 - 1.25x_3 + 0.833x_4 - 37.5x_5 \leq -1.132 \quad (12)$$

Boundary:

$$0.4 \leq x_1 \leq 0.6, 70 \leq x_2 \leq 110, 0.4 \leq x_3 \leq 0.5, 0.3 \leq x_4 \leq 0.6, 0.01 \leq x_5 \leq 0.03 \quad (13)$$

After we obtain the answer using equation (8) through (13), and then calculate the solution having each maximum gains among requester, provider2 and provider3, we present those results in Fig. 2 altogether. As a result, since Total Utility of Requester+Provider3 is the biggest, the third Service Provider is selected as the provider offering the optimal service.

4 Conclusions

In this paper, we start from web service selection problem through functional requirement without consideration of quality. Then we propose Web Service Quality Broker Architecture, which helps service requester find service provider offering the maximum gains in the requester's point of view and bind that dynamically. Negotiation process of Quality Broker is described by Multi-Attribute Utility Theory (MAUT) on the basis of quality information of both sides participating in negotiation.

The optimal web service selection method through negotiation of Quality Broker proposed in this paper can give a solution to handle the calculation complexity increment problem, which can be generated by the extension to multilateral negotiations and the reliability problem for the result value, and can be extended and applied to the optimal service selection through other negotiation strategies in the future.

References

1. Ovum, "Web Services Market Overview", Ovum Research Report, Sept, (2002)
2. Mike Clark, "UDDI weather report", Nov, (2001), Available online: http://www.webservicesarchitect.com/content/articles/clark04.asp
3. K.H.Bennett, and others, "A Broker Architecture for Integrating Data Using a Web Services Environment", ICSOC, Vol.2910, (2003) 409-422
4. Hung, P.C.K, "Web Services Discovery Based on the Trade-off between Quality and Cost of Service: A Token-based Approach", in the ACM SIGecom Exchanges, Vol. 4.2, Sept, (2003), 20-26
5. Asit Dan, Heiko Ludwig, Giovanni Pacifici, "Web Services Differentiation with Service Level Agreements", White Paper, IBM Corporation, May, (2003)
6. Keller, A., Ludwig, H., "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", Journal of Network and Systems Management, Special Issue on E-Business Management, Vol.11, No.1, Plenum Publishing Corporation, Mar, (2003)
7. Hung, P.C.K, "A Primitive Study of Logrolling in e-Negotiation", Proceedings of the 36th Annual Hawaii International Conference on, Jan, (2003), 29-36
8. Keeney, R., "Decision Analysis: An Overview", Operations Research, Vol.30, No.5, (1982), 803-838
9. Sanghyun Park, Sung-Bong Yang, "Mediator Agent System for Reciprocity and Negotiation using Multi-Attributes", Journal of KISS: Software and Applications, Vol.31, No.3, Mar, (2004), 308-316
10. M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller, "A Concept for QoS Integration in Web Services", Fourth International Conference on Web Information Systems Engineering Workshops, Dec, (2003), 149-155
11. Tao Yu, Kwei-Jay Lin, "The Design of QoS Broker Algorithms for QoS-Capable Web Services", IEEE International Conference on the e-Technology, e-Commerce and e-Service, Mar, (2004), 17-24
12. Shuping Ran, "A Model for Web Services Discovery With QoS", ACM SIGecom Exchanges, Vol.4, Issue.1, (2003) 1-10

13. NCA, "A Study on Technical Trends and Deployment Strategies of Web Service Quality Management", National Computerization Agency Research Report, Dec, (2003), Available online: http://www.nca.or.kr/eindex.htm

CA-Ex: A Tuning-Incremental Methodology for Communication Architectures in Embedded Systems[1]

Haili Wang, Jinian Bian, Yawen Niu, Kun Tong, Yunfeng Wang

Department of Computer Science and Technology, Tsinghua University, Beijing, 10084, P.R.China
{Whl01, Nyw03, Tk02, Wangyf00}@mails.tsinghua.edu.cn, Bianjn@tsinghua.edu.cn

Abstract. The *communication architecture* (CA) problem is at the very heart of system level design related to the development of distributed embedded systems. The design of efficient CAs is pivotal because communication is becoming the most important source of on-chip desired performance numbers. In this paper we focus on the aspects of CA design in heterogeneous systems consisting of arbitrarily linked multi-components, and introduce a new design methodology named CA-Ex which enables a tuning-incremental architecture exploration. Unlike previous research efforts, CA-Ex employs three kinds of optimization strategies to implement topology, mapping and scheduling scheme, and interface circuits. One of the major contributions is that we summarily present four architecting scenarios and outline a unified framework to achieve a specification-modeling-exploration process. Finally, we evaluate CA-Ex through an illustrative case study on JPEG decoder and describe its advantages.

1 Introduction

Designing distributed embedded systems is an error-prone and time-consuming process because of complicated interactions during hardware/software codesign and strict performance and cost requirements. The heterogeneous systems, such as digital television, set-top boxes, mobile terminals, are usually composed of programmable processors, off-the-shelf application-specific components and various types of interconnected communication architectures. With the rapidly increasing computation and communication power in embedded systems owed to manufacturing technology, designers rely more and more on automatic design tools and sound methodologies that allow them to explore a large amount of design solutions at the system level [1].

One of the key problems in embedded systems is the architecting of communication infrastructure for quickly exploring alternative solutions. It serves as a middle connecting link between the preceding algorithm-level and the following implementa-

[1] This work is supported in part by National Natural Science Foundation of China under grant NSFC-90207017, NSFC-60236020, NSFC-60236011 and NSFC-60121120706, and Hi-Tech Research & Development (863) Program of China under grant 2003AA115110.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 74-80, 2005.

tion-level, as illustrated in Fig.1. The problem in CA design that designers want to solve is to find the best system architecture, including automatic generation of interconnect topologies, appropriate mapping of functionality tasks and communication channels, and optimal determination of communication protocols [2][3].

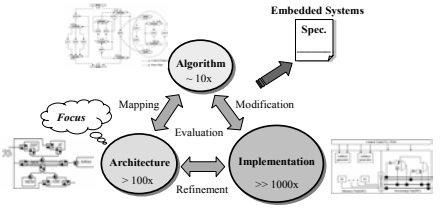


Fig. 1. Three design abstraction levels

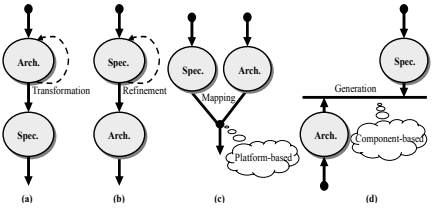


Fig. 2. Four scenarios of architecting

A large body of work focuses on CA design. There exist mostly two categories of design approaches: one is based on dynamic simulation technique of the entire system to increase the accuracy; the other is based on static analysis and evaluation for trade-offs between design time and efficiency. Lahiri et al. presented an exploration technique called CAT (Communication Architecture Tuners) by utilizing two phases of performance-analysis methodology to explore CA [3]. Renner and Glesner introduced performance modeling, generation and optimization approach of communication infrastructure for architecture-precise rapid prototyping of real-time embedded systems [4]. Eles et al. focused on the aspects related to the synthesis of distributed systems and carefully studied on the impact of scheduling with bus access optimization [5]. Based on a specific hierarchical class library [6], Zhu developed a related modeling framework for on-chip architecture and integrate this into a simulation environment. In [7], Russell and Jacome addressed an architecture-level performance evaluation, which adapts to component-based embedded systems by the use of a designer-specified scenario to support early space exploration.

Compared to the existing approaches above, our contributions are as following: 1) we summarily present four architecting scenarios and outline a unified framework to achieve architecture exploration. 2) We utilize the advantages of static analysis and dynamic simulation technique to trade-off accuracy and efficiency. 3) We propose a tuning-incremental methodology to support communication architecting.

The remainder of the paper is organized as follows. Section II introduces four architecting scenarios and our related methods. Section III describes our CA-Ex methodology for communication architecture, followed by an illustrative case study on the JPEG decoder in Section IV. The conclusion is given in Section V.

2 Architecting Scenarios and Our Methods

As mentioned previously, on-chip communication architecture has a major impact on performance in the design of heterogeneous systems, which can accommodate different components that communicate using an appropriate communication manner.

As shown in Fig.2 (which is an update of the similar figure shown in [8]), there are generally four design scenarios for the system architecting that implement the desired specification. The first one as shown in Fig.2 (a) is to satisfy the constraints on system and find an optimal architecture by a few of defined transformation rules for a given specification. The second one, as illustrated Fig.2 (b), is a reverse procedure that gradually refines the system specification until adapting to a given architecture. Fig.2 (c) and Fig.2 (d) show two popular approaches (platform- /component-based) that adopt an implementation-independent idea between the specification and architecture. The way enables both of them to develop and design respectively, and reduce design cycle time. In our methodology, we give more attention to both (b) and (c) methods and combine their advantages to implement architecture design.

Building the system architecture based on on-chip communication will raise the challenges of efficient mapping from algorithm to architecture due to the essential distinction between the two-level models, as shown in Fig.3. The algorithm model is an untimed and technology-independent description, while the architecture model is a timed and implementation-ready representation. Therefore, the direct mapping is an unclear and difficult task. Introducing an intermediate process called transaction-level will be a preferable practice, which can deal with the existing problems resulting from the representation style and behavior difference between them. Fig.3 illustrates basic elements for creating a realistic functional and architectural model. For instance, there are five types of components to generate a virtual prototype at transaction level. It is a simple task to achieve the transformation of algorithm into transaction model, followed by a smooth mapping from the transaction to architecture. Our methodology adopts this method as the front-end of architecting design.

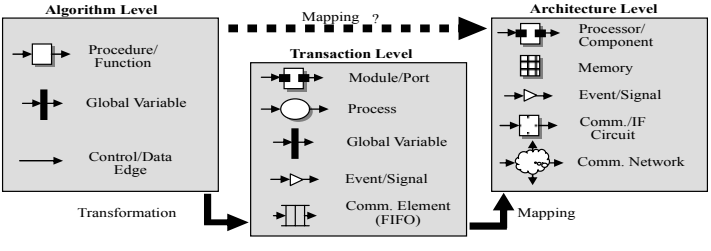


Fig. 3. Mapping from algorithm to architecture

3 CA-Ex Methodology

In this section, we present our new architectural design methodology called CA-Ex (**C**ommunication **A**rchitecture **Ex**ploration) that enables designers to easily explore the architecture-level space and achieve a specification-modeling-exploration process. CA-Ex provides a unified framework and divides the whole design flow into five sub-processes, as depicted in Fig.4. One of the advantages by the use of this methodology is that it can directly and accurately reflect and inherit design constraints (such as delay) during the mapping from function to architecture, as well as generate application-specific communication architectures in an incremental manner.

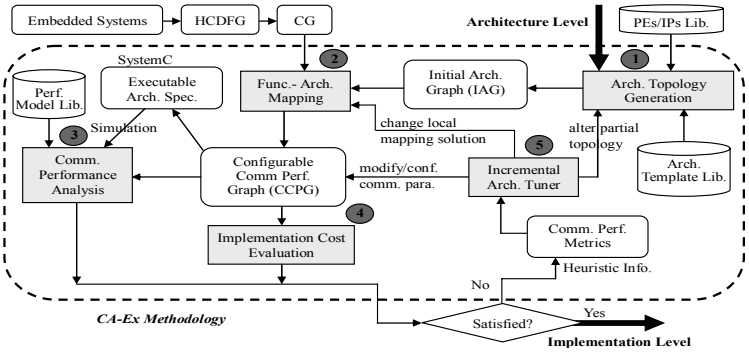


Fig. 4. CA-Ex system design flow diagram

3.1 Functional and Architectural Model

Before introducing CA-Ex Methodology, we first give the definition of communication graph (CG), which is used as the functional description of system in transaction level. CG = {V, E, T, IO} is a directed cyclic graph, where v represents each task in system behaviors. A relation edge $e_{ij} = (v_i, v_j) \in E$ indicates that data will be transferred from v_i to v_j. To each e_{ij}, we assign $t_{ij} \in T$ to depict the times of data exchanging between v_i and v_j. To separate computation from communication and enable reuse, we have constructed a virtual-component communication model to describe data transmission on a given architecture [8]. The model is denoted as a three-tuple set: {VCI, VP, VC}, where VCI, VP, and VC represent virtual component interface, virtual port, and virtual channel, respectively. The model encapsulates the implementation details of communication. Unlike previous work mentioned, we take the interface mechanism into account and model it as IO = {VCI, VP, VC}. It is useful for multi-component design because communication is a must to be considered.

The distributed architecture model of embedded systems is featured with multiple heterogeneous processing elements (PE) connected through a network of communication channels (CH). In design flow, nodes in CAG are mapped to PEs, while edges to CHs. In addition, a technology library is built to deposit the attributes of available PE and Ch types. For example, the chip area and price are associated with each PE type, while the attributes in each CH type include the chip area/price per port and transfer speed in correspondence with interface types. Note that interface type represents the communication protocol and port arrangement of a CH, which should be compatible with PEs it wants to connect. If it is not, an interface transducer will be inserted to smooth the communication. Based on the component and technology libraries, heterogeneous systems can be composed of arbitrarily linked programmable processors and off-the-shelf application specific components.

3.2 Design Flow

As mentioned above, the architecting is an activity embedded in a system development process. The activity in our CA-Ex methodology requires two types of inputs: 1)

A partitioned golden HCDFG (Hierarchical CDFG), which is a representation model for the specification of embedded systems after HW/SW partitioning; 2) A template-based platform; 3) Functional description of system at the transaction-level.

To efficiently find an optimal application-specific architecture, it is required that the derived architecture model can be analyzed and evaluated in a quick time from a large number of possible solutions. In CA-Ex, the first step has to start with an abstract architecture model that captures the required system resources. A template-based technique is used to generate an *initial architecture graph* (IAG). By obtaining the results after partitioning, and based on CG, the mapping of system behaviors to the IAG will be executed, followed by a *configurable communication performance graph* (CCPG) which can be generated. The requirements of communication behavior can be planned to the CCPG according to the result of executable transaction-level model. One of key features is to adopt the combination method of static analysis and dynamic simulation to tradeoff accuracy and efficiency. And then, the CCPG model is analyzed to evaluate and determine candidate communication architectures. Our goal is to derive a worst case delay and cost by which the system completes its execution, such that this delay is as small as possible and the cost is guaranteed to generate the sound architecture topology by optimizing protocol parameters of the communication between processes, or changing local mapping solution.

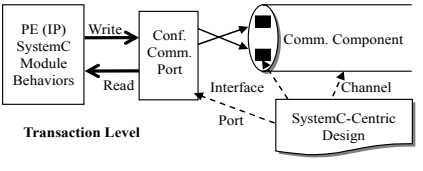


Fig. 5. SystemC-based architecture model

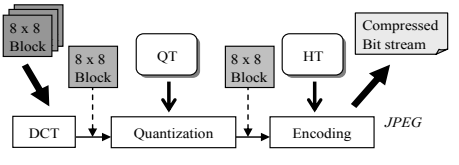


Fig. 6. JPEG image compression

In order to support efficient architecture exploration, CA-Ex introduces a popular method using an idea of the platform-based design. The architecture model has been constructed by SystemC 2.0 [9]. As shown in Fig.5, the models are defined and parameterized using performance metrics. Thus, performance analysis based on the CCPG model can be incorporated with SystemC 2.0 simulation kernel, and the analysis results will be back-annotated to the CCPG. An important characteristic in architecture exploration is that an incremental process has been introduced. We employ three tuning strategies on performance critical paths instead of globally replacing one candidate from the architecture library. These strategies include: a) Alter partial topologies; b) Change local mapping solutions; c) Configure or modify communication protocol parameters. In addition, considering the application-specific characteristics, the design space can be pruned to generate an optimal architecture.

4 Case Study: JPEG Decoder Application

To implement an efficient exploration and evaluate CA-Ex methodology, we choose a JPEG application as a testbench. Fig.6 shows a basic design flow of JPEG encoder

algorithm. In JPEG compression algorithm, there are mainly three processes: Discrete Cosine Transform (DCT), Quantization and Huffman code. In the succeeding case study, we focus on the JPEG decoder that is a reverse transformation process.

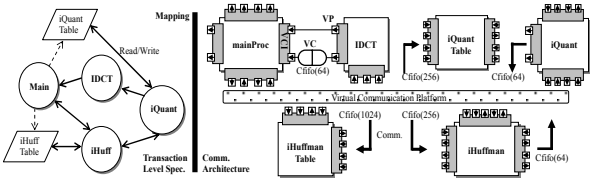


Fig. 7. Transaction-level modeling

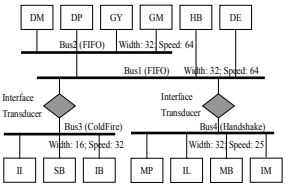


Fig. 8. Topology exploration

As discussed previously, we introduce the transaction-level modeling to smooth the mapping process between algorithm and architecture. The left of Fig.7 illustrates an example of JPEG decoder modeled as a corresponding CG in transaction level discussed above. Each node represents a set of tasks to implement a given behavior, such as IDCT that reverts to a real image from an equivalent in the frequency domain. The communication among nodes can be achieved by encapsulated read and write functions using SystemC master/slave library. The right of Fig.7 illustrates an architecture model with six modules, which are all connected by custom bus but only the IDCT is connected with the *mainProc* module by the point-to-point manner. The modeling results are summarized in Table 1. It can be seen from the table that two types of communication manner are used in this architecture. The Cfifo is viewed as channel FIFO and the number followed by it is the depth of FIFO.

Table 1. Results of communication architecture exploration

Source Node	Link Node	Comm. Manner	Source Node	Link Node	Comm. Manner
iQuant	*IDCT*	Cfifo(64)	*mainProc*	*iHuff*	Cfifo(256)
	iHuff	Signal(1)		*iQuant table*	Cfifo(256)
	iQuant table	Signal(1)		*iHuff table*	Cfifo(1024)
iHuff	*mainProc*	Signal(1)	*iQuant table*	*iQuant*	Signal(1)
	iQuant	Cfifo(64)	*iHuff table*	*iHuff*	Signal(1)
	iHuff table	Signal(1)	*IDCT*	*mainProc*	Cfifo(64)

The outputs of architectural exploration by using CA-Ex include: golden topology, mapping and scheduling scheme, performance statistics, and interface circuits. Until now, we have achieved the automatic generation of interconnect topology. The other three parts of results are not implemented completely and under investigation.

Taking the CG above as input, we apply the channel mapping process to generate the fine-granularity architecture for JPEG decoder (for exploring more space in architecture level, we recombine different tasks in JPEG decoder), shown in figure 8. The diamonds in the final CA represent interface transducers for smoothing communication between PEs with incompatible protocols. The results of topology exploration can be also seen from the figure that four different types (such as width, speed, and communication protocol) of CA mechanism (buses) are used in this architecture.

5 Conclusions

In this paper, we have presented a novel architecting design methodology called CA-Ex, to assist designers to explore early architecture space for distributed embedded systems. We also illustrate design scenarios and related issues in the architecture-level for achieving a specification-modeling-exploration process, and propose an overall design flow to solve them. We have used the CA-Ex methodology for a case study of JPEG decoder application. Experimental results conducted to evaluate the effectiveness of the unified framework indicate that the methodology can perform well and implement space exploration.

Future work will focus on investigating the intrinsic relation between algorithm development and architecture design, and smoothing the integration of architecting and low-level layout.

References

1. V. D. Zivkovic, P. Lieverse: An Overview of Methodologies and Tools in the Field of System-level Design. Lecture Notes in Computer Science, Vol. 2268. Springer-Verlag, Berlin Heidelberg New York (2002) 74-89
2. J-P. Calvez, V. Perrier: SOC Architecting and Design with CoFluent Studio, Concepts and Methodology -Part I-. available at: http://www.cofluent.com
3. K. Lahiri, A. Raghunathan, S. Dey: Design Space Exploration for Optimizing On-Chip Communication Architecture. In IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 23, No. 6, pp. 952-961, June 2004.
4. F-M. Renner, J. Becker, M. Glesner: Automated Communication Synthesis for Architecture-precise Rapid Prototyping of Real-Time Embedded Systems. In IEEE International Workshop on Rapid System Prototyping, pp. 154-159, 2000.
5. P. Eles, A. Doboli, P. Pop, Z. B. Peng: Scheduling with Bus Access Optimization for Distributed Embedded Systems. In IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol.8, No.5, pp. 472-491, Oct. 2000.
6. X. P. Zhu, S. Malik: A Hierarchical Modeling Framework for On-Chip Communication Architecture. In IEEE/ACM International Conference on Computer Aided Design, pp. 663-670, Nov. 2002.
7. J. T. Russell, M. F. Jacome: Architecture-Level Performance Evaluation of Component-Based Embedded Systems. In Proc. of the 40th Design Automation Conference, pp. 394-401, 2003.
8. Haili Wang, Qiang Wu, Jinian Bian et al.: A Novel Virtual-Real Component Synthesis Approach in SoC Design. In the 8th International Conference on CAD/Graphics03, Macau, pp. 151-156, Oct. 2003.
9. Open SystemC Initiative (OSCI). available at: http://www.systemc.org

Efficient Parallel Spatial Join Processing Method in a Shared-Nothing Database Cluster System

Warnill Chung[1], Soon-Young Park[2], and Hae-Young Bae[2]

[1]Electronics and Telecommunications Research Institute,
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea
wnchung@etri.re.kr
[2]Dept. of Computer Science & Information Engineering, INHA University,
253 YongHyun-3Dong, Nam-Gu, Inchon, 402-751, Korea
{sypark, hybae}@dblab.inha.ac.kr

Abstract. Spatial database cluster consisted of several single nodes on high-speed network to offer high-performance is raised. But, research about spatial join operation that can reduce the performance of whole system in case process at single node is not achieved. Therefore, we propose efficient parallel spatial join processing method in a spatial database cluster system that uses data partitions and replications method that considers the characteristics of spatial data. Since proposed method does not need the creation step and the assignment step of tasks, and additional message transmission between cluster nodes that appear in existent parallel spatial join method, it shows performance improvement of 23% than the conventional parallel R-tree spatial join for a shared-nothing architecture about expensive spatial join queries.

1 Introduction

Cluster systems have been studied widely. However, researches about cluster system for spatial data are insufficient. But, it is very inefficient that division method of data and redundancy method that is used in existing spatial database cluster system apply to spatial data of bulk that expensive CPU operation that regional adjacency is high [3,8,9,11]. Spatial database cluster manages spatial data at each cluster node by dividing spatial data into spatial relations, and uses partial replication method that replicates regional relation [1,7]. Spatial join query is achieved with parallel at all cluster nodes that manage redundancies of join relation and each node executes spatial join operation about space objects of logical division area allocated to own node. Result tuples of spatial join operation performed at each node are transmitted to query processing node by pipelining. Query processing node transmits query results that are transmitted from other nodes to user without removing redundancy result separately. In proposed spatial database cluster system, parallel spatial join method does not need work generation and assign step between each node those are need to achieve parallel

*This research was supported by University IT Research Center Project.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 81-87, 2005.

spatial join [10]. In the midst of parallel spatial join operation, it processes expensive spatial join query rapidly because there is no message transmission cost for work division between cluster nodes. Assignment of logical division area shortens the query processing time because redundancy of disk I/O for equal object between nodes is no need. Also, several nodes do not achieve expensive refinement operation repeatedly at filter step of spatial join operation by filtering method that uses central point of MBR of object in the logical division area border. And response time is fast because relevant results are transmitted to user who demands query immediately without achieving additional work of union and so on to remove duplicate results about parallel space join results. Hardware platform of spatial database cluster for our research is based on a shared-nothing structure that is consisted of several independent workstations and services stably large data [5].

2 Related Works

Spatial join operation is to seek a set of object pair that satisfied specification spatial condition that is contain, cross, and overlap etc for two set of spatial data. And it has characteristic that operation time increases rapidly according as the number of object increases by requiring multiplex injection for two sets of data [3]. Previous research about these spatial join was spatial join that uses mostly single processor, and actuality spatial join's performance was progressed rather. But, in spite of this research, spatial join at single processor is not satisfying users who require fast response time. The reason is that the numbers of object to be spatial join's target is much and operation of each object is complicated and it takes much unit operation time [2]. Many researches for parallel spatial join [2,3,4,6] have been studied to overcome limit in this single processor by using multi processor. Among them, there is a research about distributed parallel spatial join that makes two nodes participate spatial join parallel under distributed spatial database system environment [8]. Spatial join characteristic is that target object is a lot and expense of refinement step in operation of object pair costs much. This separates and runs operation for each object pair [6]. Also, it provides advantage on performance to divide into filter step and refinement step [3]. Both a point that it is no relativity in operation of this object pair and a point that can perform by multistage are very profitable characteristic inside parallel.

3 Parallel Spatial Join Scheme

3.1 Determination of Cluster Node to Achieve Join

Cluster node that must process spatial query gets list of cluster nodes to run spatial join query parallel through position information of local relation managed in global meta-information. These nodes transmit query of parse tree form and own node ID to each nodes without special data transmission and request spatial join operation because those stores relevant relation already. Each cluster nodes that are required spa-

tial join operation achievement run spatial join operation about spatial objects in relevant area after those find out spatial join's area to perform by oneself by referring logical partition area of join target relations in own local meta information. For example, let's suppose that replicated spatial relation R and S in node 1, node 2 and node 3 join as following condition. Following join query is separated to following three queries by LPA that is LPA_Node1, LPA_ Node2 and LPA_ Node3.

SELECT * FROM R, S WHERE CONTAINS(R.Obj, S.Obj);

Each cluster node that is requested spatial join operation achievement runs spatial join operation about spatial objects that overlapped in own logical partition area. After that, it transmits result records to node that requests spatial join operation.

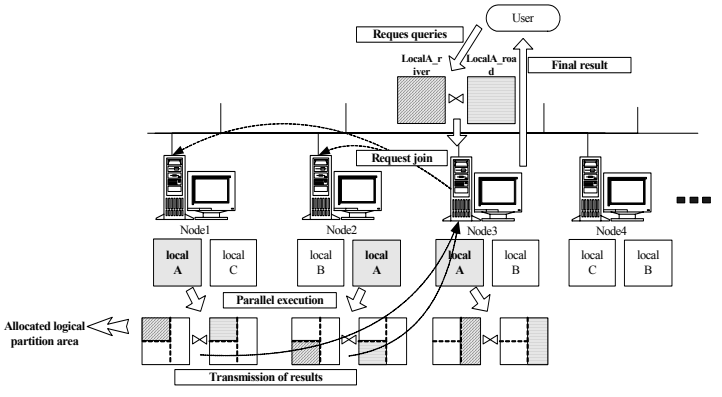


Fig. 1 Parallel spatial join achievement by using logical partition area

Fig. 1 is an example of parallel spatial join achievement that uses logical partition area about spatial join query for the river and road layer in local area A. Query processing node that receives query from user transmits user query to relevant nodes and achieves parallel join operation after it searches nodes that store join target relations repeatedly by referring global meta information. Nodes that obtain spatial join operation achievement achieve spatial join operation about spatial objects that are included in logical partition area that is established to own local meta information. Spatial join's results outputted parallel at each node are passed to query processing node and the query processing node transmits final results to user.

3.2 Replicated Candidate Object Elimination Method at Filtering Step

By performing filtering step using CPM(center point of MBRs) of object, our method prevents achieving expensive refinement operations that occur repeatedly about spatial objects in partition area boundary. Local node that processes query achieves spatial join operation about objects overlapped with logical division area of own local meta-information. However, if parallel spatial join operation is based on space partitioning, expensive spatial join operation is executed repeatedly like Fig. 2. By taking advantage of filtering technique which uses center point of spatial object's MBR, expensive replicated join operation which occurs due to objects in partition area

boundary is removed. All tuples of spatial relation have spatial header information to approach spatial object fast. This header information has MBRs and CPM. By filtering spatial objects that exist in logical partition area boundary line according to CPM, replicated expensive refinement operation is prevented in several nodes.

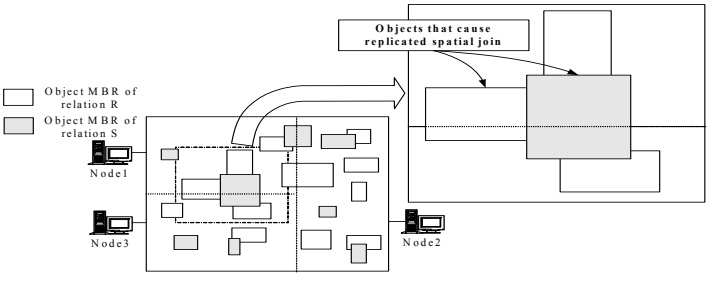


Fig. 2 Spatial objects by replicated join

Fig. 3 shows spatial join operation about spatial relation A and B stored replica in cluster nodes. Spatial object A1 and B1 exist on boundary line of NODE1_LPA and NODE2_LPA. Filtering step determines standard relation and then produces final candidate object pair according to standard relation and CPM of candidate object. As doing like this, replicated refinement operation that is caused due to replicated candidate object pairs is prevented.

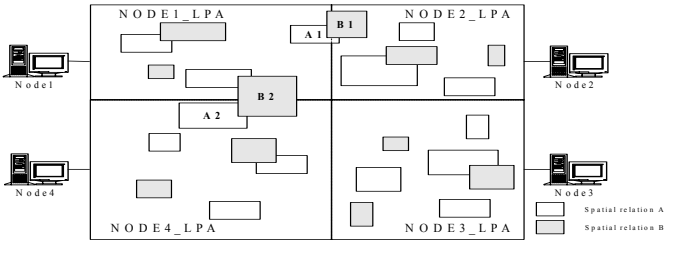


Fig. 3 Spatial join operation of spatial relation A and B

Fig. 4 shows these cases. LPA is logical partition area and A is MBR of spatial object within standard relation.

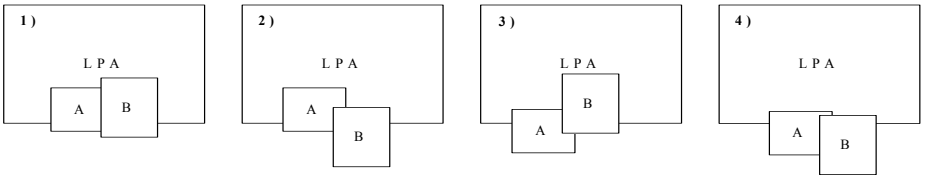


Fig. 4 The case being candidate object pair on LPA boundary

Like 1) and 2) in Fig. 4, in case CPM of spatial object in standard relation is in own logical partition area, refinement operation is achieved as outputting final candidate object pair. In case of 3) and 4), Node which manages logical partition area under relevant area is achieved. However, like Fig. 5(3), spatial objects that are included perfectly in other logical partition area can be omitted in filtering step.

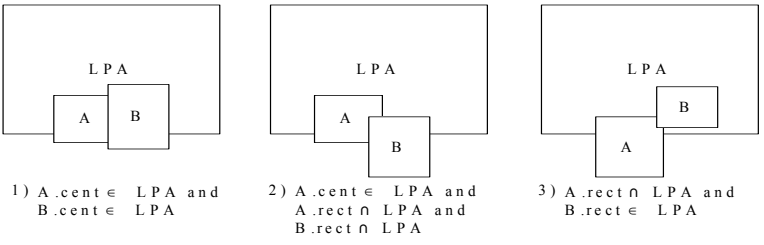


Fig. 5 The case being candidate objects on LPA boundary

Fig. 5 shows the case that outputs final candidate object pair in cluster node that has LPA's logical partition area and conditional expression. Filtering operation uses R tree that is constructed in all spatial relations. Filtering algorithm is based on R-tree that uses replicated candidate object elimination method.

4 Performance Evaluations

The proposed method and conventional method [4] were implemented in shared-nothing spatial database cluster system [7]. R-tree index constructed in all relations that are used in estimation and filtering step is achieved by using R trees in spatial join. Also, transmission time that was spent to transmit relevant results to user was except in this performance estimation because that time is equal in all methods.

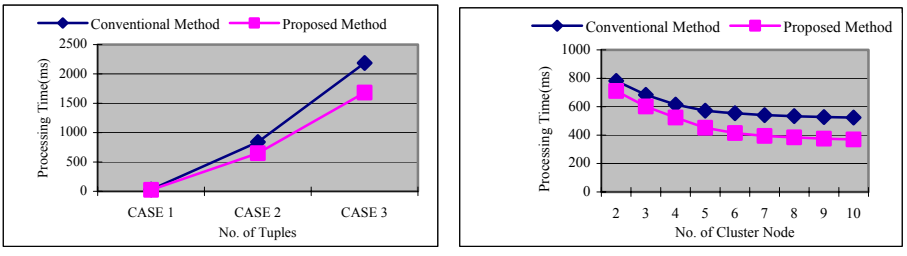
Table 1 is information about spatial relations that are used in this experiment. Spatial join achievement used crossing operation, and the number of cluster node to take part in parallel spatial join operation is four.

Table 1. Detail inforamtion of realtions for evaluations

	Relation name	Relation size(Kb)	No. of Tuples
CASE 1	Subway	42	112
	HangjungDongGae	80	100
CASE 2	River	498	1,220
	Dong-KyunGae	607	587
CASE 3	Building	1,348	7,739
	Road	1,394	6,482

Firstly, we analyze the query processing time according to size of spatial join target relation. Fig. 6(a) shows that proposed method achieves faster query processing than existing method according to size increment of relation. This shows that proposed method is more efficient. The reason is because this method achieves spatial

join operation parallel at all nodes by using logical partition area assigned in each node without process of work creation and so on.



(a) Size of Relation (b) Number of Cluster Nodes

Fig. 6 Variations of parallel spatial join performance

Secondly, we analyze the query processing time according to increment of the number of cluster node. Spatial relation that is used in estimation is the river of CASE 2 and Dong-KyungGae and spatial join operation is crossing operation. Fig. 6(b) shows that processing time of proposed method are decreased greatly than existing techniques according to increment of cluster node's number. According to increment of cluster node, this method has no transmission of message between nodes and equal Disk I/O frequency by using logical partition area based on MBR that considers area adjacency of spatial data.

5 Conclusions

With the rapid growth of the Internet, significant numbers of web-based information systems have come to rely on database cluster technology to serve large user communities and to deal with peak loads. Therefore, we proposed a parallel spatial join method to process expensive spatial join operation efficiently by using partition method and replication method of spatial data in spatial database cluster system in terms of high throughputs, fast response time, data consistency, linear scalability and fault-tolerance. In our method, there is no need task creation and allocation of conventional method because each cluster node achieves spatial join operation about spatial objects in LPA allocated to oneself. So, expensive spatial join queries can be processed rapidly because there is not necessary message transmission cost for task distribution between cluster nodes.

References

1. B. Kemme, Database Replication for Clusters of Workstations, Ph.D. thesis, Department of Computer Science, ETH Zurich, Switzerland, 2000.
2. J.D. Kim, and et. al, A Study on Task Allocation of Parallel Spatial Joins using Fixed Grids, KIPS Journal Vol.8-D NO.04 pp.347~360, 2001.

3. T. Brinkhoff and H.P. Kriegel, Parallel Processing of Spatial Joins Using R-trees, Proceedings of 12th Int'l Conf. on Data Eng.(ICDE'96), New Orleans, LA, 1996.
4. L. Mutenda, et. al, Parallel R-tree Spatial Join for a Shared-Nothing Architecture, 1999 Int'l Symposium on Database Applications, pp. 429-436, 1999.
5. Y.I Jang, et. al, Web GIS Cluster Design with Extended Workload-Aware Request Distribution Strategy, Proc. of KISS, VOL. 28, NO.02, pp.304 ~306, 2001.
6. Y.D. Seo, Implementation and Performance Evaluation of Parallel Spatial Join Algorithm using R-tree, Master Thesis, Pusan National Univ., 1999.
7. Chung-Ho Lee, A Partial Replication Protocol and a Dynamically Scaling Method for Database Cluster Systems, Ph.D Thesis, Inha Univ., 2003.
8. H.J. Lee, Parallel Pipelined Spatial Join Method for Efficient Query Processing In Distributed Spatial Database Systems, Master Thesis, Inha Univ., 2002.
9. C.G. Li, Load Balancing Method using Proximity of Query Region in Web GIS Clustering System, Master Thesis, Inha Univ., 2001.
10. Jignesh M. Patel and David J. Dewitt, Partition Based Spatial-Merge Join, Proc. of ACM SIGMOD, Vol.25, Issue 2, pp. 259-270, 1996.
11. K. Tamura, and et. al, The Parallel Processing of Spatial Selection for Very Large Geo-Spatial Databases, ICPADS 2001, pp. 26-30, 2001.

Maximizing Parallelism for Non-uniform Dependence Loops Using Two Parallel Region Partitioning Method

Sam Jin Jeong

Division of Information and Communication Engineering, Cheonan University
Anseo-dong 115, Cheonan City, Korea 330-704
sjjeong@cheonan.ac.kr

Abstract The existing parallelizing compilers can parallelize most of the loops with uniform dependences, but they do not satisfactorily handle loops with non-uniform dependences. Most of the time, the compiler leaves such loops running sequentially. Unfortunately, loops with non-uniform dependences are not so uncommon in the real world. This paper presents the two parallel region partitioning method of nested loops with non-uniform dependences for maximizing parallelism. By parallelizing anti dependence region using variable renaming, we will divide the iteration space into two parallel regions by a line in case that FDT (Flow Dependence Tail set) does not overlap FDH (Flow Dependence Head set). Comparison with some related works shows more parallelism than other existing methods.

1 Introduction

Given a sequential program, a challenging problem for parallelizing compilers is to detect maximum parallelism. It is generally agreed upon, and shown in the study by Kuck and et al. [1] that most of the computation time is spent in loops. Therefore, current parallelizing compilers pay much of their attention to loop parallelization. A loop can be easily parallelized if there are no cross-iteration dependences. But loops with cross-iteration dependences are very common in normal programs.

Some techniques, based on Convex Hull theory [5] that has been proven to have enough information to handle non-uniform dependences, are the minimum dependence distance tiling method [4], the unique set oriented partitioning method [3], and the three region partitioning method [7].

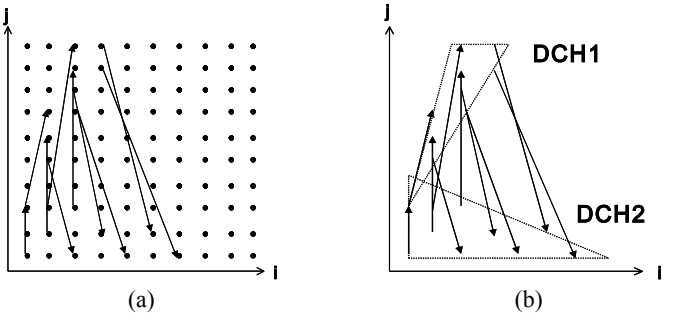
Fig. 1(a) shows the dependence patterns of Example 1 in the iteration space.

Example l.

```
do i = 1, 10
    do j = 1, 10
        A(2*i+3, j+1) = . . .
              . . . = A(i+j+3, i+2*j+1)
    enddo
enddo
```

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 88-93, 2005.

This paper will focus on parallelization of flow and anti dependence loops with non-uniform dependences. Especially, it shows us a case that the iteration space is divided into two parallel regions by a line.



(a) (b)

Fig. 1. (a) Iteration Spaces (b) CDCH of Example 1

The rest of this paper is organized as follows. Chapter two describes our loop model, and introduces the concept of Complete Dependence Convex Hull (CDCH). In chapter three, we define the properties of FDT (Flow Dependence Tail set) and FDH (Flow Dependence Head set). We show how to find FDT and FDH and to divide iteration space into two parallel regions by a line. Chapter four shows comparison with related works. Finally, we conclude in chapter five with the direction to enhance this work.

2 Program Model and Dependence Analysis

We consider doubly nested loop program of the form shown in Fig. 2. For the given loop, l_1 (l_2) and u_1 (u_2) indicate the lower and upper bounds respectively, and should be known at compile time. We also assume that the program statements inside these nested loops are simple assignment statements of arrays. The dimensionality of these arrays is assumed to be equal to the nested loop depth. To characterize the coupled array subscripts, the array subscripts, $f_1(I, J)$, $f_2(I, J)$, $f_3(I, J)$, and $f_4(I, J)$, are linear functions of the loop index variables.

```
do I = l1, u1
    do J = l2, u2
          A(f1(I, J), f2(I, J)) =   .  .  .
                                .  .  .  = A(f3(I, J), f4(I, J))
    enddo
enddo
```

Fig. 2. A doubly nested loop model

The loop in Fig. 2 carries cross iteration dependences if and only if there exist four integers (i_1, j_1, i_2, j_2) satisfying the system of Linear Diophantine Equations given by (1) and the system of inequalities given by (2). The general solution to these equations

can be computed by the extended GCD or the power test algorithm [6] and forms a **DCH** (Dependence Convex Hull).

$$f_1(i_1, j_1) = f_3(i_2, j_2) \text{ and } f_2(i_1, j_1) = f_4(i_2, j_2) \quad (1)$$

$$l_1 \le i_1, i_2 \le u_1 \text{ and } l_2 \le j_1, j_2 \le u_2 \quad (2)$$

From (1), (i_1, j_1, i_2, j_2) can be represented as

$$(i_1, j_1, i_2, j_2) = (g_1(i_2, j_2), g_2(i_2, j_2), g_3(i_1, j_1), g_4(i_1, j_1))$$

where g_i are linear functions.

From (2), two sets of inequalities can be written as

$$l_1 \le i_1 \le u_1 \text{ and } l_2 \le j_1 \le u_2 \text{ and} \quad (3)$$
$$l_1 \le g_3(i_1, j_1) \le u_1 \text{ and } l_2 \le g_4(i_1, j_1) \le u_2$$

$$l_1 \le i_2 \le u_1 \text{ and } l_2 \le j_2 \le u_2 \text{ and} \quad (4)$$
$$l_1 \le g_1(i_2, j_2) \le u_1 \text{ and } l_2 \le g_2(i_2, j_2) \le u_2$$

And, (3) and (4) form DCHs denoted by DCH1 and DCH2, respectively. Clearly, if we have a solution (i_1, j_1) in DCH1, we must have a solution (i_2, j_2) in DCH2, because they are derived from the same set of equations (1). The union of DCH1 and DCH2 is called Complete DCH (**CDCH**), and all dependences lie within the CDCH. Fig. 1(a) shows the CDCH of Example 1, which is given in [3].

If iteration (i_2, j_2) is dependent on iteration (i_1, j_1), then we have a dependence vector $d(i_1, j_1) = (d_i(i_1, j_1), d_j(i_1, j_1)) = (i_2 - i_1, j_2 - j_1)$

So, for DCH1, we have

$$d_i(i_1, j_1) = g_3(i_1, j_1) - i_1 = (a_{11} - 1)i_1 + \beta_{11} j_1 + \gamma_{11} \text{ and} \quad (5)$$
$$d_j(i_1, j_1) = g_4(i_1, j_1) - j_1 = a_{12} i_1 + (\beta_{12} - 1) j_1 + \gamma_{12}$$

For DCH2, we have

$$d_i(i_2, j_2) = i_2 - g_1(i_2, j_2) = (1 - \alpha_{21}) i_2 - \beta_{21} j_2 - \gamma_{21} \text{ and} \quad (6)$$
$$d_j(i_2, j_2) = j_2 - g_2(i_2, j_2) = -a_{22} i_2 + (1 - \beta_{22}) j_2 - \gamma_{22}$$

The properties of DCH1 and DCH2 can be found in [3].

3 Two Parallel Region Partitioning Method

We define the flow dependence tail set (FDT) and the flow dependence head set (FDH) as follows. We can form two regions, FDT and FDH, by the algorithm, which is similar to the algorithm presented in [5]. Fig. 3(a) shows the head and tail sets of flow dependence, anti dependence in Example 1.

Definition 1 *Let L be a doubly nested loop with the form in Fig. 2. If line* $d_i(i_1, j_1) = 0$ *intersects DCH1, the flow dependence tail set of the DCH1, namely* ***FDT****(L), is the region H, where H is equal to*

$$\text{DCH1} \cap \{(i_1, j_1) \mid d_i(i_1, j_1) \geq 0 \text{ or } d_i(i_1, j_1) \leq 0\} \quad (7)$$

Definition 2 *Let L be a doubly nested loop with the form in Fig. 2. If line* $d_i(i_2, j_2) = 0$ *intersects DCH2, the flow dependence head set of the DCH2, namely* ***FDH****(L), is the region H, where H is equal to*

$$\text{DCH2} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) \geq 0 \text{ or } d_i(i_2, j_2) \leq 0\} \quad (8)$$

Property 1 *Suppose line* $d_i(i, j) = p*i+q*j+r$ *passes through CDCH. If* $q > 0$, *FDT(FDH) is on the side of* $d_i(i_1, j_1) \geq 0$ ($d_i(i_2, j_2) \geq 0$), *otherwise, FDT(FDH) is on the side of* $d_i(i_1, j_1) \leq 0$ ($d_i(i_2, j_2) \leq 0$).

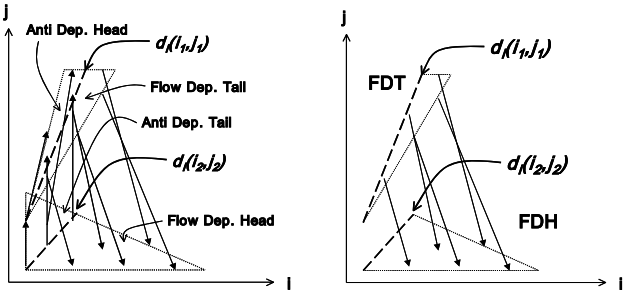


Fig. 3. (a) The head and tail sets of flow dependence, anti dependence, (b) FDT and FDH in Example 1.

By Property 1, we can know the area of the flow dependence head set (FDH) of DCH1 and the flow dependence tail set (FDT) of DCH2 in Example 1 as shown in Fig. 3(b).

Because the intersection of FDT and FDH is empty, FDT does not overlap FDH and the iteration space is divided into two parallel regions by the line $d_i(i_2, j_2) = 0$. From equation (6), we can get $d_i(i_2, j_2) = i_2/2 - j_2/2$, and the equation is $j = i$. So, the iteration space is divided into two parallel regions, AREA1 and AREA2, by the line $j = i$. The execution order is AREA1 → AREA2.

Transformed loops are given as follows.

```
/* AREA1 */
 do i = l1, u1
      do j = max(l2, ⌈i⌉), u2
           A(2*i+3, j+1) = . . .
             . . . = A(i+j+3, i+2*j+1)
      enddo
 enddo
```

```
/* AREA2 */
do i = l1, u1
    do j = l2, min(u2, ⌈i⌉)
        A(2*i+3, j+1) = . . .
          . . . = A(i+j+3, i+2*j+1)
    enddo
enddo
```

4 Performance Analysis

Theoretical speedup for performance analysis can be computed as follows. Ignoring the synchronization, scheduling and variable renaming overheads, and assuming an unlimited number of processors, each partition can be executed in one time step. Hence, the total time of execution is equal to the number of parallel regions, N_p, plus the number of sequential iterations, N_s. Generally, speedup is represented by the ratio of total sequential execution time to the execution time on parallel computer system as follows:

$$Speedup = (N_i * N_j)/(N_p + N_s) \quad (9)$$

where N_i, N_j are the size of loop *i, j,* respectively.

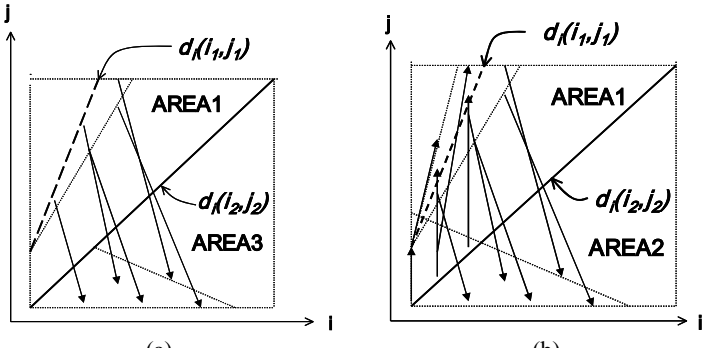


Fig. 4. Regions of the loop partitioned by (a) the three region partitioning; (b) the unique sets oriented partitioning in Example 1

In Example 1, the three region partitioning method [2], [7] divides the iteration space into one parallel region, AREA3, and one serial region, AREA1, as shown in Fig. 4(a). So, the speedup is (10*10)/(1+45) = 2.2.

The unique set oriented partitioning method [3] divides the iteration space into one parallel region, AREA2, and one serial region, AREA1, as shown in Fig. 4(b). So, the speedup is the same as the three region partitioning method.

Applying the minimum dependence distance tiling [4], $d_{jmin} = 2$. The space can be tiled with width = 1 and height = 2, thus 50 tiles are obtained. The speedup for this method is (10*10)/(50) = 2.

Our proposed method divides the iteration space into two parallel areas as shown in Fig. 3(b). The speedup for this method is (10*10)/2 = 50.

5 Conclusions

In this paper, we have studied the parallelization of nested loops with non-uniform dependences to maximize parallelism, and proposed Two Parallel Region Partitioning Method.

When there are both flow and anti dependence sets, we eliminate anti dependence from the doubly nested loop by variable renaming. After variable renaming, there remains only flow dependence in the nested loop. We then divide the iteration space into the flow dependence head and tail sets.

If FDT does not overlap FDH, a line between two sets divides the iteration space into two parallel areas by our proposed method.

In comparison with some previous partitioning methods, our proposed methods give much better speedup than other methods.

References

1. D. Kuck, A. Sameh, R. Cytron, A. Polychronopoulos, G. Lee, T. McDaniel, B. Leasure, C. Beckman, J. Davies, and C. Kruskal, "The effects of program restructuring, algorithm change and architecture choice on program performance," in *Proceedings of the 1984 International Conference on Parallel Processing*, August 1984.
2. C. K. Cho and M. H. Lee, "A loop parallization method for nested loops with non-uniform dependences", in *Proceedings of the International Conference on Parallel and Distributed Systems,* pp. 314-321, December 10-13, 1997.
3. J. Ju and V. Chaudhary, "Unique sets oriented partitioning of nested loops with non-uniform dependences," in *Proceedings of International Conference on Parallel Processing*, vol. III, pp. 45-52, 1996.
4. S. Punyamurtula and V. Chaudhary, "Minimum dependence distance tiling of nested loops with non-uniform dependences," in *Proceedings of Symposium on Parallel and Distributed Processing*, pp. 74-81, 1994.
5. T. Tzen and L. Ni, "Dependence uniformization: A loop parallelization technique," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 5, pp. 547-558. May 1993.
6. M. Wolfe and C. W. Tseng, "The power test for data dependence," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 5, pp. 591-601, September 1992.
7. A. Zaafrani and M. R. Ito, "Parallel region execution of loops with irregular dependences," in *Proceedings of the International Conference on Parallel Processing*, vol. II, pp. 11-19, 1994.

The KODAMA Methodology: An Agent-Based Distributed Approach

Guoqiang Zhong, Satoshi Amamiya, Kenichi Takahashi, and Makoto Amamiya

Graduate School of Information Science and Electrical Engineering,
Kyushu University, 6-1, Kasuga-Koen, Kasuga, 816-8580 Japan
{zhong, roger, tkenichi, amamiya}@al.is.kyushu-u.ac.jp

Abstract. The KODAMA methodology is our endeavour to explore new analysis and design methodologies, as well as new tools, for developing ubiquitous applications around autonomous, interacting software agents. To concrete and detail the well-known multiagent system paradigm, KODAMA introduces a plug-and-play agent model, an agent community model and an on-demand interaction model. At the top level, a whole system is decomposed into various agent communities. Working one level down, communities are broken down into independent agents. At the lowest level, agent roles are the basic entities for specifying agent activities in online interactions. In this article, we first present how these new models are exploited in the analysis and design phases; then discuss some details of how they are implemented in a practical shopping-support system.

1 Introduction

In an era of *ubiquitous computing* that Mark Weiser foresaw in [1], the new software engineering challenges that must be faced are characterised by three key features. First, most systems are now de facto concurrent and distributed, and are expected to interact in flexible ways to dynamically exploit services and make context-dependent decisions that can not be foreseen at design time. Second, more and more systems are moving towards a customer-centred paradigm, in which many aspects of customer behaviour should be facilitated. Third, and as a natural consequence of the first two features, systems have to be designed as open systems so that new components may join (and existing components may leave) the system constantly, and interactions may occur at unpredictable times, for unpredictable reasons, between unpredictable components [2].

Against this background, the KODAMA (Kyushu University Open & Distributed Autonomous Multiagent) methodology exploits the *multiagent system* (MAS) paradigm to provide a set of new analysis and design models for developing ubiquitous applications. From the micro level to the macro level, KODAMA introduces a plug-and-play agent model, an agent community model and an on-demand interaction model. When taken together, these new models form a complete mapping between the characteristics of complex systems and the key

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 94–102, 2005.

abstractions necessary for modelling agents, agent organisations and agent interactions. These abstractions in turn serve as clear guidelines for the full system development and maintenance cycle.

Different from other methodologies such as Gaia [3] and ARCHON [4], our approach offers a holistic methodology and software engineering guidance on (i) how to conceptualise and structure applications as agent-based systems, (ii) how to explicitly represent agent social relationships to dynamically form, maintain, and disband organisations, (iii) how to distribute different tasks throughout the community. The emphasis of KODAMA therefore is on the smooth integration of analysis, design, development and maintenance of multiagent systems.

The remainder of this article is organised as follows. Section 2 details the KODAMA methodology as it pertains to agent-oriented software engineering (AOSE). Section 3 examines a case study of a shopping-support system to show some details of its implementation. We then outline, in Section 4, some concluding remarks.

2 The KODAMA Methodology

The core concepts behind agent-oriented software engineering, in effect, are agent-oriented decomposition, agent-oriented abstraction and agent-oriented organisation [5,6].

Agent-oriented decomposition: the problem-solving space of a system is naturally partitioned into self-contained agents.

Agent-oriented abstraction: agents, either cooperating or competing with each other, are the right metaphor for both analysing and modelling subsystems, subsystem components, interactions, and organisation relationships.

Agent-oriented organisation: by their very nature, agents are social entities not only in the sense that they need to interact, but in the sense that they rely on others according to protocols determined by their roles.

In the following two subsections and Section 3, we detail how these three concepts are exploited in KODAMA for the analysis, design and implementation of multiagent systems.

2.1 The Analysis Phase

The main goal of the analysis phase is to collect and specify the requirements of the system-to-be and to identify some generic characteristics of the system that are likely to remain the same regardless of the actual applications.

Just as the real world can be viewed as a set of autonomous agents that collaborate to perform some higher-level function [7], agent-oriented decompositions based on functions/actions/processes are more intuitive and closer to the way people might think about a problem [8]. Accordingly, the KODAMA approach defines a set of preliminary models of agents, interactions and organisations as follows:

The preliminary agent model. Basically, agents have full control over their behaviour and states. An agent, for example, can only ask other agents to help it by sending them messages. The operating engine of an agent is message-driven. Furthermore, an agent's social knowledge is clearly separated from its domain-level knowledge so that the application designer can reuse many common facilities provided by the former and focus on the latter.

The preliminary interaction model. This model distinguishes agent interaction activities into message exchanging, message interpretation and inter-role interaction. It is worth noting that data exchange activities are application independent and can be insulated from agent-related issues. Meanwhile, the format and syntax of messages are system-neutral and specified by declarative agent communication languages (ACL), while the semantics of messages are determined by agent roles in applications and specified by interaction protocols. Situated in an environment which is subject to ongoing change, it is more realistic for agents to have the ability to initiate (and respond to) interactions in a flexible manner so that they can respond to unanticipated difficulty by sending requests for assistance at runtime.

The preliminary organisation model. This model captures the dependencies and interrelationships between various agents in the system in terms of organisation rules and organisation structures. On the one hand, organisation rules allow designers to explicitly define how new groups of agents can be formed, how unwanted groups of agents can be disbanded. On the other hand, organisations of different sizes may require different organisational structures from a simple collection of peers to a relatively complex multilevel hierarchy. When an agent interacts with others in the system, the notion of which agents are primitive can be varied according to its aims and objectives. From a different level of abstraction, a collection of agents may be viewed as a single conceptual unit. Agent-based systems always involve ever-changing interrelationships among their members that are explicitly represented by organisational structures, and organisation rules that are explicitly defined by interaction protocols.

2.2 The Design Phase

Starting with the three preliminary models outlined in the analysis phase, the design phase is responsible for eventually elaborating them into real models, which in turn act as guidelines for actual system implementations.

The plug-and-play agent model. This plug-and-play standard mandates that an agent is made up of a *kernel unit*, which encapsulates the common modules, and an *application unit*, which encapsulates application-dependent modules. In practice, the kernel unit is composed of a set of common data and modules, such as agent ID, registers, control module, communication module, etc; the application unit is composed of one or more *plug-in* modules, which can be seamlessly plugged into the kernel unit.

As mentioned above, agents are inherently message-driven. When a new message reaches an agent, it is first checked and interpreted by the kernel unit. Then the application-dependent part of the message is forwarded to and processed in

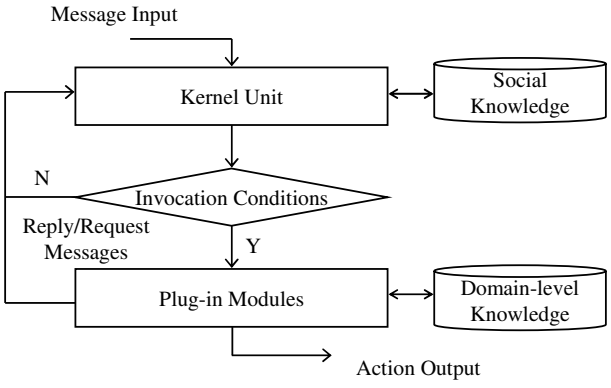


Fig. 1. The message flow in an agent

the application unit, which may result some action being performed, replying to the message or sending a new request message(s) (see Figure 1). Typically, a plug-in module consists of invocation condition, which is checked with incoming messages, and plug-in program, which is called if the condition is satisfied.

To install a plug-in module, either during development or runtime, an agent simply adds the invocation condition to the kernel unit and loads the plug-in program to the application unit. Similarly, an agent can uninstall a plug-in module by removing its invocation condition and plug-in program. Like class libraries in object-oriented programming, plug-in libraries in our agent-oriented approach are modular and reusable.

The agent community model. All agents in KODAMA are categorised into various *agent communities* (or communities for short), which in turn are linked together to form organisation structures. It seems appropriate to divide complex systems into smaller more manageable sub-systems, each of which can be represented and handled by one or more agent communities. Social relationships between agents therefore are specified through their positions within communities. Determined by the size and needs of applications, organisation structures can be peer-centred or hierarchical, and can even change dynamically in an evolutionary fashion.

Our model for communities, namely the *portal agent* model, is based on three inter-related concepts: *role*, *interaction* and *organisation*. At the micro level, roles are the basic building blocks representing generic agent behaviours that can interact mutually. At the macro level, an organisation is composed of a set of role-playing agents and their interactions. Agents, on the one hand, are specified as active communicative entities which play roles. The behaviour of a multiagent system as a whole, on the other hand, is the result of roles played by agents.

A portal agent acts only on behalf of a single community and allows all agents in the community to be treated as one single normal agent, the portal agent itself, outside the community. In general, an agent's address in a community, we call

it *logical address*, is given by the community name and its unique identifier as follows:

```
<logical-address> ::= <community-name> ‘‘.’’ <UID>
<community-name>  ::= <logical-address> | ‘‘root’’
<UID>             ::= <string>
```

A community name actually is the logical address of the portal agent that is given by the higher community. Note that the portal agent of the top community is called *root*.

The on-demand interaction model. The KODAMA methodology defines a *push* and *pull* model for online interaction and cooperation between agents. With this model, agent attributes are divided into those belonging to a *public profile*, which is open and can be made public, and those belonging to a *private profile*, which is restricted and cannot be exposed. Agent roles are offered through online interaction between agents by pushing their public profiles to service agents and pulling available roles and rules back. Those roles are in turn evaluated locally according to private profiles and corresponding rules. It is worth noting that the notion of agent roles here is consistent with the notion of plug-in modules mentioned earlier. While the plug-in model focuses on abstraction mechanisms for individual agents, the role model captures agent activities in interactions.

Policy packages are used to pack together roles (representing agent services), assignment rules of roles (which are described in a rule base), and service contents [9]. The structure of policy packages is as follows:

```
<policy package> ::= <rules> <roles> <contents>
<rules>          ::= <rule> | <rule> <rules>
<rule>           ::= <condition> <role names>
<role names>     ::= <role name> | <role name> <role names>
<condition>      ::= ‘‘TRUE’’
                   | ‘‘and’’ <condition> <condition>
                   | ‘‘not’’ <condition>
                   | ‘‘eq’’ <attribute>
                   | ‘‘<’’ <attribute>
                   | ‘‘>’’ <attribute>
<attribute>      ::= <variable name> <value>
<roles>          ::= <role> | <role> <roles>
<role>           ::= <role name> <programme name> <init description>
<contents>       ::= <content> | <content> <contents>
<content>        ::= <programme name> <programme path>
```

Public and private profiles are stored as pairs of variable and value, or as digital certificates with *Public Key Infrastructure* (PKI). They can only be accessed by a specific *profile manager*. Meanwhile, policy packages are evaluated by a specific *policy evaluator*. Once an agent gets a policy package, its policy evaluator gets roles by evaluating rules in the policy package. In this way, agent roles are described and delivered as policy packages; agent attributes are managed as public and private profiles; appropriate roles are extracted through a two-step

Table 1. Agents, roles and functions

agents	application independent roles / application dependent roles	functions
portal agent	portal	(de-)register agents, filter messages
	agent list delivery	send shop agent list
creator agent	create agent	create new agents
	delete agent	delete agents
	join community	join an agent community
	information registration	register visitor & shop information
shop agent	join community	join an agent community
	leave community	leave an agent community
	information registration	register shop information
	policy package delivery	send shop policy packages
user agent	join community	join an agent community
	leave community	leave an agent community
	location	update location, join community
	profile management	manage public/private profiles
	package evaluation	evaluate shop policy packages
	advertising	send advertising email to visitors

matching process: first matching against the public profile on the service provider agent side, then matching against the private profile on the service consumer agent side.

3 A Case Study

As a part of an academe-industry joint research project, we were able to build a shopping-support system and perform a large-scale experiment [10] in the Osu shopping mall in Nagoya, Japan. Through this case study, we demonstrate how our approach can be used to build a location- and context-aware application in which participants, activities and transactions are treated as agents, agent roles and agent interactions respectively.

The actual system is developed in Java language and its implementation proceeds from two different perspectives simultaneously. One is a top-down approach, looking at the application's overall structure, and the other is a bottom-up approach, deciding the granularity of the agents and determining each agent's roles in the community. In particular, we chose a single-level hierarchical structure for the agent organisation with four kinds of agents, two kinds of agent communities (see Figure 2). Furthermore, agents' roles are decomposed, in accordance with the plug-and-play agent model and on-demand interaction model, into application-independent roles, and application-dependent roles, as summarised in Table 1.

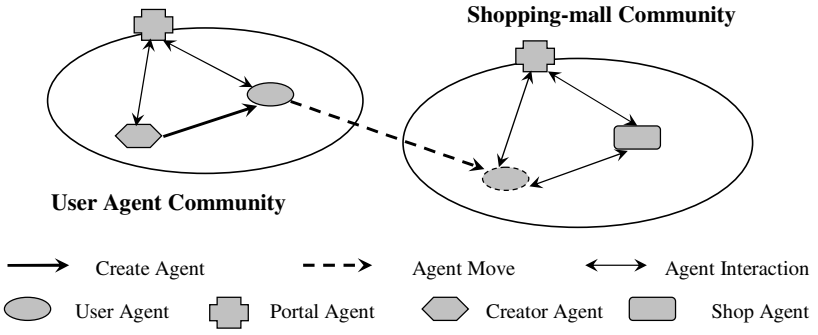


Fig. 2. An overview of the agent society

Table 2. Definition of the location role

role name:	location
protocol name:	shopping-support system
description:	update location
initiator:	location sensing subsystem
interaction with:	join community role/local portal agent
input:	position information
parameters:	shopping-mall name
output:	invoke join community role/shop agent list enquiry
parameters:	portal agent name/position information

To demonstrate the working mechanism of agent roles, here we present one specific application-dependent role, the *location* role. As shown in Table 2, the location role is initiated externally by a location-sensing subsystem (refer [10] for details). Depending on its position information, the agent has two choices: either to move to another shopping-mall community or to update the shop agent list. In the former case, the location role will initiate the join community role in the agent. In the latter case, the location role will interact with the local portal agent to get a new shop agent list.

This experimental system development was divided into three steps: common facilities development, kernel unit development and application unit development. The greatest effort was devoted to the application unit part and required six person-months to fix the specification and three person-months to program. The development of both the common facility part and the kernel unit part, however, was based on our previous work and required approximately another two person-months. As illustrated in Table 3, the common facility part and the kernel unit part make up 47% of the source code, while the application unit part makes up 53% of the source code. It means that the KODAMA approach promotes speedy development cycle and high reusability.

Table 3. Source code constitution of the experimental system

	number of classes	lines of code	percent
common facility	50	3,756	21%
kernel unit	63	4,646	26%
application unit	153	9,547	53%
total	266	17,949	100%

4 Conclusions

This paper has given a general introduction to the KODAMA methodology for agent-oriented software engineering. In sum, our approach affords four benefits. First, the role-agent-community metaphors provide the underlying rationale for the system under analysis and guide subsequent design, development and deployment. Second, KODAMA is naturally distributed and capable of self-configuration and self-adaptation. Both service providers and consumers are encapsulated as relatively independent agents. Third, the agent community model and portal agent model are suitable not only for partitioning a complex system into smaller sub-systems, but also for integrating the agent level and the macro level seamlessly. Fourth, the public profile and private profile management model, together with the push and pull ad hoc interaction model, enables on-demand interaction and extracts customised services among agents.

References

1. Weiser, M.: The computer for the 21st Century. Scientific American **265** (1991) 94–104
2. Papazoglou, M.P.: Agent-oriented technology in support of e-business. Communications of the ACM **44** (2001) 71–77
3. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: the Gaia methodology. ACM Transactions on Software Engineering and Methodology **12** (2003) 317–370
4. Jennings, N.R., Mamdani, E.H., Corera, J.M., Laresgoiti, I., Perriollat, F., Skarek, P., Varga, L.Z.: Using Archon to Develop Real-World DAI Applications. IEEE Expert **11** (1996) 64–70
5. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons, Chichester England (2002)
6. Luck, M., McBurney, P., Preist, C.: Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing). Technical report, AgentLink (2003)
7. Jennings, N.R.: Agent-based control systems. IEEE Control Systems Magazine **23** (2003) 61–73
8. Booch, G.: Object-Oriented Analysis and Design with Applications. Addison Wesley (1994)

9. Iwao, T., Okada, M., Kawashima, K., Matsumura, S., Kanda, H., Sakamoto, S., Kainuma, T., Amamiya, M.: Large Scale Peer-to-Peer Experiments with Virtual Private Community (VPC) Framework. In: Proceedings of Cooperative Information Agents. LNAI 2442, Springer Verlag (2002) 66–81
10. Zhong, G., Amamiya, S., Takahashi, K., Iwao, T., Kawashima, K., Ishiguro, T., Kainuma, T., Amamiya, M.: You've Got Mail From Your Agent. In: Engineering Societies in the AgentsWorld IV. LNAI 3071, Springer Verlag (2004) 392–409

A New Iris Recognition Approach for Embedded System

Hongying Gu[1], Yueting Zhuang[1], Yunhe Pan[1], and Bo Chen[2]

[1] Institute of Artificial Intelligence, Zhejiang University,
Hangzhou 310027, P.R.China
{guhy,yzhuang}@cs.zju.edu.cn
[2] Software College, Zhejiang University of Technology,
Hangzhou 310032, P.R.China
cb@zjut.edu.cn

Abstract. Iris recognition is a prosperous biometric method, but some technical difficulties still exist especially when applied in embedded systems. Support Vector Machine (SVM) has drawn great interests recently as one of the best classifiers in machine learning. In this paper, we develop an iris recognition system using SVM to classify the acquired features series. Even though the SVM outperforms most of other classifiers, it works slowly, which may hinder its application in embedded systems, where resources are usually limited. To make the SVM more applicable in embedded systems, we make several optimizations, including Active Learning, Kernel Selection and Negative Samples Reuse Strategy. Experimental data show that the method presented is amenable: the speed is 5 times faster and the correct recognition rate is almost the same as the basic SVM. This work makes iris recognition more feasible in embedded systems. Also, the optimized SVM can be widely applied in other similar fields.

1 Introduction

Embedded systems have limited resources, including power, communications bandwidth, time and memory. All of the above will require new ways of thinking, not just at the input and output ends, but about the very fundamentals of computing and communications. Ways will be needed to ensure such systems to operate reliably, safely, and efficiently. Support Vector Machine (SVM) has been a promising method for classification because of its solid mathematical foundations [1]. In classifying the same features series, it works better than most of other classifiers, such as Euclidean distance and Hamming distance. Nevertheless, due to its nature of the computational complexity, under certain circumstances, it may not be applicable in embedded systems.

Iris recognition is one of the best biometrics recognition methods in the world [2]. The combination of the iris recognition and embedded systems would surely create great possibilities. An embedded iris recognition system can be used in cell phones, automobiles, e-business systems and so on. Recently, health researchers

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 103–109, 2005.

are investigating microscopic sensors that could traverse the bloodstream, monitor health conditions and report them tirelessly. Consumer electronics and information technology companies envision homes filled with intelligent devices that can interact with each other. They also envision homeowners and appliance manufacturers to improve the quality of daily life. In these embedded applications, it is necessary that the identification be recognized. Since the cameras are already embedded into cell phones and other places for other purposes, iris recognition is firstly considered to recognize a person.

There are several companies developing iris recognition products: Iridian Technologies, LG and Panasonic. Among the products provided by different companies, the BM-ET300 from Panasonic is unique in that it is partly embedded iris recognition product. All of these companies use Daugman's iris recognition algorithm and most of their products run on PC or similar platform. In Daugman's algorithm [3], multi-scale Gabor filters are used to demodulate texture phase structure information of the iris to get an IrisCode. The difference between a pair of IrisCode is measured by their distance.

In this study, we choose variation fractal dimension as the feature of irises according to the self-similarity of the rich texture variation details in irises. To be more suitable in embedded applications, we improve the efficiency of SVM for iris recognition and maintain its high classification performance at the same time. The experimental data show that our iris recognition system with an optimized SVM is 5 times faster than the system with the normal SVM while the accuracy is almost the same.

2 Iris Recognition

As a typical pattern recognition method, iris recognition is done in 2 steps. First, extract features to present iris pattern. Then, choose a classifier to do the pattern recognition. In the features extraction, we choose variation fractal dimensions as features because of the self-similarity of irises textures. Later, we will discuss two different classifiers and compare their performance in the section of Experiments and Results.

2.1 Features Extraction

By analyzing local intensity variations of an iris image, we can see the self-similarity of the local variations of the two-dimensional iris image. So we take it as a typical fractal phenomenon. The mathematical way to measure fractals is by fractal dimension. Mandelbrot [4] studied the use of some effective fractal dimensions. The most well known and the most widely used one is box-counting fractal dimension. Inspired by Mandelbrot, A special box-counting dimension was proposed for texture images like irises in [5]:

$$Dim_B F = \lim_{\delta \to 0} \frac{\log N_\delta(IsChange(F_\delta))}{-\log \delta}, \tag{1}$$

where $N_\delta\left(IsChange\left(F_\delta\right)\right)$ is the smallest number of square boxes of side δ necessary to cover F. F_δ is the image, which covered by square boxes of side δ, $IsChange(F_\delta)$ is 1 if the gray scale in F_δ changes, and 0 otherwise.

In order not to leave out any variation details, we use a moving window (l_a, l_b) which moves by $(Step)$. The overlap is allowed when covering the image. We now calculate the variation fractal dimension for every sub-image.

$$Feature(I) = \left(D_1 \cdots D_n\right), \tag{2}$$

where D_i is i^{th} sub-image's corresponding variation fractal dimension. These features are sent to the classifier for recognition.

2.2 Match by Euclidean Distance

After extracting the features, we are apt to go straight forward to measure the distance between different people's feature series. Both of [2] and [3] use distance as their classifier to do iris recognition. One is Hamming Distance and the other is Euclidean Distance. Hamming distance is 1-D distance, and Euclidean distance is 2-D.They are both linear classifiers and time-saving algorithms. Here we use Euclidean Distance to do the comparison experiment. Suppose we have two feature vectors: x and y, in Euclidean space, the distance is defined as:

$$d\left(x, y\right) = \sqrt{\sum_{i=1}^{n} \left(x_i - y_i\right)^2}. \tag{3}$$

Euclidean distance works fast and performances well when comparing linear features. But iris textures are highly nonlinear, and the variation fractal dimension features are nonlinear too. Therefore, the conventional Euclidean Distance is not suitable to differentiate people represented by variation fractal dimension. The possible solution is to construct a nonlinear classifier so that the two classes can be correctly classified.

2.3 SVM Classifier

Based on the minimization of structural risk of statistical learning theory, SVM works like this [6]: it maps the input vector x to a higher dimension feature space Z, and constructs a classification-hyperplane in this space. The hyperplane H is $w \cdot x + b = 0$.

The following is the general equation of the SVM decision function for classification:

$$f\left(x, \alpha\right) = \operatorname{sgn}\left(\sum_{SupportVectors} y_i \alpha_i k\left(x_i, x\right) + b\right), \tag{4}$$

where $y_i\alpha_i = w_i$ are the networks weights, x_i are the support vectors of the solution, b is the threshold of the function and $k\left(x_i, x\right)$ is the kernel function.

As we can see, the solution is the sign of the addition, so this is the generalization function for two-class's classification. In our case, the kernel function is then the polynomial function of degree d: $k(x, y) = (x \cdot y + c)^d$.

3 Performances and Optimizations

How to build an effective learning system plays a crucial role in the performance of classifiers. Support Vector Machine (SVM) is a promising method for classification because of its solid mathematical foundations which convey several salient properties that other methods hardly provide. However, the efficiency of training support vector machines is still a bottleneck, especially for a large-scale learning problem [7]. Therefore, it is important to develop a fast training algorithm for SVM in order to solve various engineering problems. The main task of this paper is to improve its efficiency.

3.1 Active Learning

Support Vector Machine has got significant success in numerous real-world learning tasks. However, like most machine learning algorithms, it is generally applied using a randomly selected training set classified in advance.

Since iris recognition system can have a database before test, we try a different strategy: choosing the training samples beforehand. Pre-setting training samples is a kind of active learning [8].

Given an unlabelled pool U, an active learner l has three components: (f, q, X). The first component is a classifier, $f : X \to \{-1, 1\}$, trained on the current set of labelled data X (and possibly unlabelled instances in U too). The second component $q(X)$ is the querying function that, given a current labelled set X, decides which instance in U to query next. The active learner can return a classifier f after each query (online learning) or after some fixed number of queries.

Usually the total training set is much larger than the number of final support vectors. Active learning can remove most non-support vectors quickly so that the computational cost for sequential optimization can be dramatically reduced. Further, another fact is that the result is changed little if some non-support vectors are removed from the training set. So we can limit the training set size, and remove the non-support feature vectors.

3.2 Kernel Selection

To get a better performance, we choose polynomial kernel function from the three kernel functions of the SVM method: Polynomial, RBF and sigmoid neural networks. The sigmoid neural network kernel function and the RBF kernel function work fine, but cost more time.

The principal parameter of the polynomial kernel function is the degree of the polynomial. It is always possible to implement with a variable degree. The polynomial degree would be between 1 and 4. Basically, we will choose the polynomial degree as 2.

Let us suppose that the iris images have a size $t_m \times t_m$ and that $t_b \times t_b$ is the block size. t_b^2 is thus the number of pixels to be processed by window of classification. If we take again the decision function of SVM with a polynomial kernel of degree d and $C = 1$:

$$f(x,\alpha) = \operatorname{sgn}\left(\sum_{SupportVectors} y_i\alpha_i\left[(x_i,x)+1\right]^d + b\right). \quad (5)$$

So the complexity of this algorithm is: $t_b^2 + d + 1$ operations by support vector.

3.3 Negative Samples Reuse Strategy

In an application of SVM, the positive samples and negative samples are non-symmetrical. Most of the cases are: negative samples are much more than positive ones and are much easier to get. Usually, when different people are tested, the negative sample set can be similar. So we reuse the negative samples when they support vectors. It improves the performance.

3.4 Other Small Optimizations

We also make some small optimizations: 1. Storing features in a dense format. The SVM must read in the training set and then perform a quadratic optimization. It takes the time to do some I/O so as to get features into the memory and process them. If we store the feature vector in a contiguous memory address, and in a dense format, it saves systems run time. 2. Saving training results. We save training results into two files. One is used to store kernel parameters, support vectors and the corresponding. The other (index file) is to store the sequential number of support vectors on the training set in order to merge them during the testing stage and reduce unnecessary kernel re-evaluations.

4 Experiments and Results

To evaluate the performance of the proposed method, we provide evidence of our analysis on SVM-based iris recognition using CASIA Iris Image Database from National Laboratory of Pattern Recognition (NLPR), Institute of Automation (IA), Chinese Academy of Sciences (CAS). The database includes 756 iris images from 108 different eyes of 80 subjects. The images are acquired during different sessions and the time interval between two collections is one month, which is a real-world application case simulation. All these experiments are done in a Pocket PC with Windows CE. The algorithms are developed in Embedded Visual C++ 3.0.

According the algorithms discussed above, we implement the iris recognition system as Figure 1:

In some real world applications, such as biometrics recognition, the reliability rate is more important than the raw error rate. So it is necessary to evaluate SVM's rejection performance. The reliability is defined by $Reliability = \frac{Recognition\ rate}{100\% - Rejection\ rate}$. Now we list the experimental result as the widely-used form shown in Table 1:

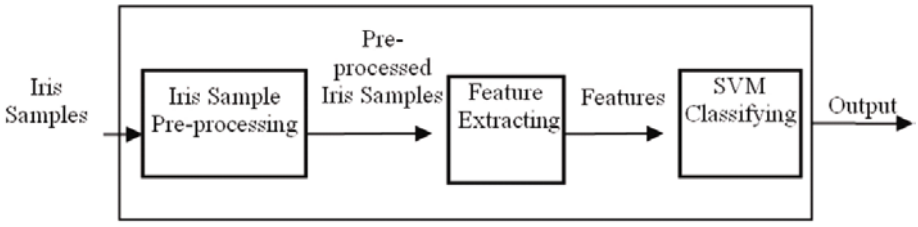


Fig. 1. Iris recognition system structure

Table 1. Comparison of CRR, FAR and FRR

Matching Methods	Recognition rate (%)	False accept rate (%)	False reject rate (%)	Reliability (%)
Euclidean Distance	72.2	17.1	10.7	80.85
SVM	98.4	0.35	1.25	99.65

Table 2. Comparison of CRR and Computational Complexity

Matching Methods	Recognition rate (%)	Average run time(s)
Euclidean Distance	72.2	0.13
SVM	98.4	4.59
Optimized SVM	98.27	0.88

As shown in Table 2, Computational Complexity comparison is done among the three different algorithms. The average run time of SVM and optimized SVM include training time and matching time.

From Table 2, we can see the optimized SVM-based iris recognition runs much faster than the original one. And its correct recognition rate is still satisfying. With this improved performance, iris recognition can be implemented in embedded systems.

5 Conclusions

The increasingly important role of embedded iris recognition system in the wide variety applications has opened an array of challenging problems centered on the computation efficiency. In this paper, we have presented an efficient approach for SVM-based iris recognition. To make the SVM more applicable in embedded systems, we make several optimizations, including active learning, kernel selection and negative samples reuse strategy. By the optimizations, the performance is improved by more than 5 times, which makes embedded iris recognition more feasible.

Support vector machine is a widely used and promising method for classification. The optimizations of the SVM we make here can be applied in a wide variety of application fields.

References

1. DeCoste, D., Scholkopf, B.: Training invariant support vector machines. Machine Learning, **46** (2002) 161-190.
2. Ma, L., Wang, Y., Tan, T.: Iris recognition based on multichannel Gabor filtering, In Proc.5th Asian Conf. Computer Vision, **I** (2002) 279-283.
3. Daugman J.G., High confidence visual recognition of persons by a test of statistical independence. IEEE Transactions on Pattern Analysis and Machine Intelligence, **36** (1993) 1148 1161.
4. Mandelbrot B. B., The Fractal Geometry of Nature, San Francisco, CA: Freeman (ed.), 1982.
5. Gu, H., Pan, H., Wu, F., Zhuang, Y., Pan, Y.: The research of iris recognition based on self-similarity. Journal of Computer-Aided Design and Computer Graphics, **16**(2004) 973-977 (in Chinese).
6. Vapnik V N.: Statistical Learning Theory, J. Wiley, New York(1998).
7. Collobert, R., Bengio, S.: SVMTorch: Support vector machines for large-scale regression problems. Journal of Machine Learning Research, **1** (2001) 143-160.
8. Tong, S., Koller, D.: Support vector machine active learning with applications to text classification, Journal of Machine Learning Research (2001) 45-66.

A RAID Controller: Software, Hardware and Embedded Platform Based on Intel IOP321

Xiao-Ming Dong, Ji-Guang Wan, Rui-Fang Liu, and Zhi-Hu Tan

Key Laboratory of Data Storage System, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, P.R.China
xmdong@mail.whut.edu.cn

Abstract. While demand for large-scale storage services is growing very rapidly, RAID is still today's standard solution for enterprise class storage systems. The software and hardware designing of a RAID controller based on Intel IQ80321 platform is introduced, in which embedded Linux is setup on the board, and a prototype system is implemented on x86 platform with Fibre Channel interface to host. The benchmark test presents throughput of 186MB/s and 184MB/s for RAID 5 reading and writing respectively.

1 Introduction

Storage demands are growing rapidly with the increasing usages of multimedia, stream casting, and large scale database. To satisfy the capacity and availability requirements of these applications, storage systems typically contain large disk arrays that are capable of storing terabytes of data and have high availability. High-performance RAID (Redundant Array of Independent Disks, [1] and [2]) storage is a critical component for many large-scale data-intensive applications.

The principle behind RAID-style architecture is simple: using a high-speed interconnection medium to aggregate arbitrarily many (slow) storage devices into a faster logical storage service, and making all applications unaware of this aggregation. RAID level 0 (stripe), 1 (mirror) and 5 (stripe and block parity check) are most commonly used.

RAID is divided into two camps: software and hardware. Software RAID uses the computer's CPU to perform RAID operations, while hardware RAID uses specialized processors. The goal of our project is to implement high performance external RAID controller based on Intel IOP80321. We also deploy our design and prototype implementation on x86 platform. With our new algorithm, the benchmark test results present throughput of 186 and 184MB/s for sequential RAID 5 read and write operations. The performance evaluation seems incredible by approaching the full bandwidth of fiber channel connection.

In the following chapters, section 2 and 3 introduce the software and hardware design. Section 4 introduces the embedded development environment. The prototype implementation is discussed in section 5, and evaluation results are also shown there. The last two sections are related works and conclusions.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 110–117, 2005.

2 Software Design

The RAID controller's software is divided into six modules, as shown in Figure 1. SCSI Target module will receive I/O requests sent from host servers passing through some type of connection, such as SCSI or Fibre Channel. In general, codes of SCSI subsystem in most operating systems (Linux, for example) are only for initiator mode operation. Thus, we must implement target SCSI drivers ourselves. On the other end, there is also a SCSI initiator module connecting to low level disk drivers and disk hardware. It must support hot plug of disks, and do detections of disk status periodically.

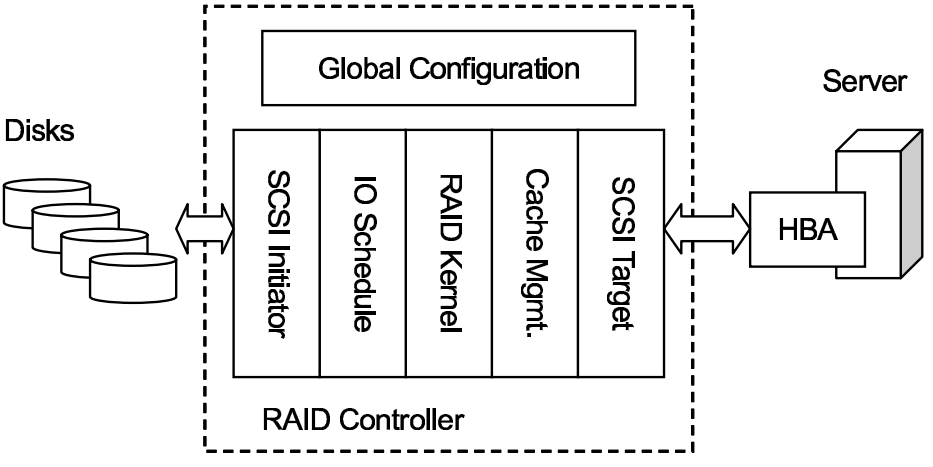


Fig. 1. This is the functional block diagram of the RAID controller software. A whole disk array consists of the RAID controller plus a group of disks (maybe FC, SCSI, or SATA type disk)

Cache Management module has a center position in the whole software system. All memory buffers are allocated at CM in size of a stripe unit, according to buddy algorithm. And the design has a "zero-copy" memory allocation feature, which means each stripe unit has only one copy in the system, avoiding the cost to copy buffers between software modules. There are two parts of cache in system for read and write respectively. We need a method to deal with all these stripe units efficiently, since operations for each I/O request are required.

Several commonly used RAID levels have been implemented in RAID Kernel module, including level 0, 1, 5, and etc. RAID algorithms transform read and write requests to a visual disk to requests to physical disks. I/O requests passed by RK will be sent into queues. I/O Schedule module is always checking the queue aiming to combine two contiguous requests into one, or do other reorganizations to enable maximum I/O throughput. Global Configuration enables

administrators to setup disk array through several approaches, such as serial consoles and custom GUI applications.

3 Hardware Scheme

The Intel IOP321 I/O processor [3], featuring an Intel XScale core (which is compliant with ARM version 5TE.) at 600 MHz, delivers a combination of high performance, PCI-X data throughput, low-power consumption, small package size, and outstanding price/performance.

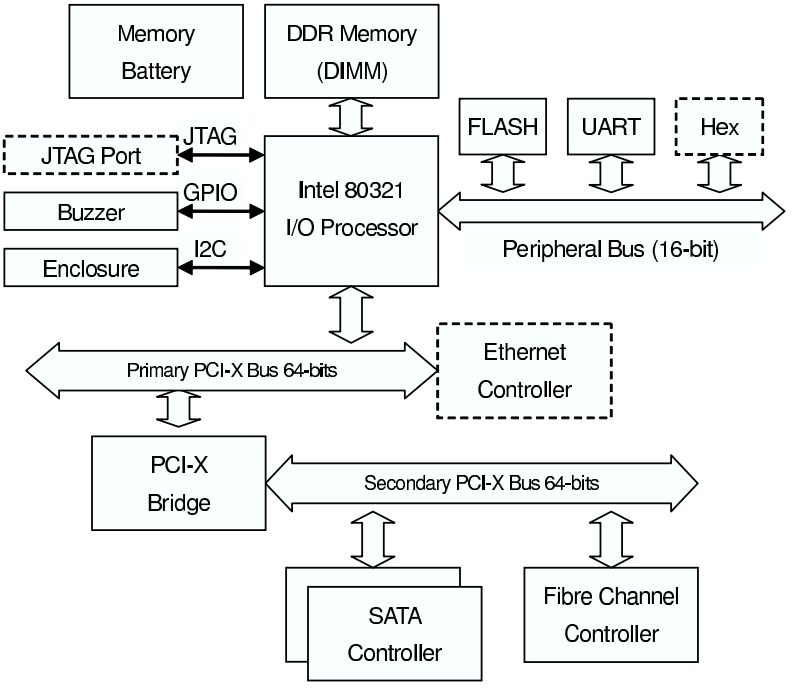


Fig. 2. This is the functional block diagram of the RAID controller hardware. Its components mainly include IOP321 processor, SATA disk controller (such as Intel 31244) and FC controller (such as Agilent Tachyon DX2 or Qlogic ISP2312)

We have designed a RAID controller based on Intel 80321 processor, as shown in Figure 2. This is a FC-to-SATA controller, with Fibre Channel interface to the host and Serial ATA interface to disks. There are many choices for SATA controller vendors, such as Silicon Image, VITESSE and Intel. Intel 31244, for example, is a 4-port PCI-X to Serial ATA Controller. We need two such chips to support at least 8 disks, providing about 2 TB of storage. It seems that there are no more choices for FC controllers other than Agilent and Qlogic, such

as Tachyon DX2 and ISP2312, respectively. Both of these are dual-port 2Gb bandwidth Fibre Channel controllers.

4 Embedded Development Platform

The target platform of the RAID controller in the system is a customized board based on Intel IOP321 that runs embedded Linux kernel. Currently, the software is developed with IQ80321 evaluation board. Features of IQ80321 board include one PCI-X expansion card slot, 8 Mbytes flash ROM, one serial console port based on the 16C550 UART, and one Intel 82544 Gbit Ethernet device [4].

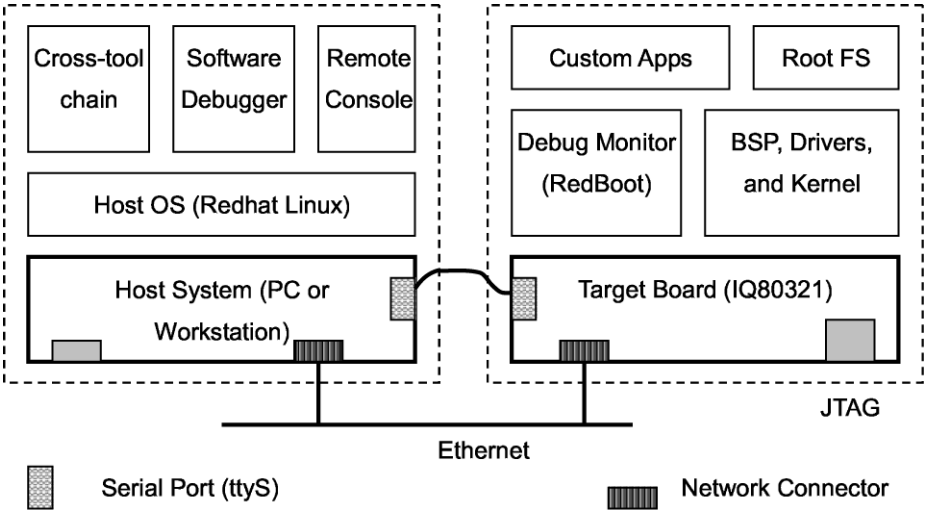


Fig. 3. The common Host/Target style cross-development environment for embedded system is shown in this figure. There are usually three types of connection between host and target platform: serial-UART, Ethernet network and JTAG debug (via an Emulator, not showed in the figure). And the above blocks show software functions

We choose Linux as the target operating system. It's a fundamental task to bring Linux startup on target board when building an embedded Linux system [5]. Based on the host/target development environment as shown in Figure 3, an embedded ARM Linux system is setup by the following 4 steps.

1. **Prepare Cross-development Toolchain:** The toolchain actually consists of a number of components, such as gcc, binutils and glibc. The first step is to compile toolchain from source packages on host side.

2. **Compile ARM-Linux Kernel:** To compile a Linux kernel for IOP321 processor, several patches should be applied to the official Linux kernel. For example, we patched rmk1, ds0, and dj9 revision packages to linux-2.4.21 kernel source tree, then compiled kernel using arm-linux toolchain, and finally delivered the kernel image file — zImage.
3. **Make Root Filesystem:** One of the last operations conducted by the Linux kernel during system startup is mounting the root filesystem. The content of a root filesystem include the system libraries (/lib), the kernel modules (/lib/modules), kernel images (/boot), device nodes (/dev), main system applications (/bin and /sbin), and custom applications (/local), with configuration files and system initialization scripts (/etc). We have made a root filesystem with size less than 2 Megabytes using BusyBox [6] utilities package.
4. **Boot the Board:** Another very important system component, bootloader, is needed in our system, which is responsible for loading the kernel during system's startup. RedBoot [7] is the standard embedded system debug/bootstrap environment from Red Hat.

5 Prototype Evaluation Results

A PC architecture RAID controller has been implemented to test algorithms and evaluate performances. It consists of an Agilent HHBA-5221A PCI-FC controller (2Gb bandwidth) acting as SCSI target, an LSI 21320-R ULTRA320 SCSI controller, and an Intel Xeon 1.80GHz CPU. The software is implemented as Linux kernel modules, including a SCSI target mode device driver for the FC card. We run Iometer [8] to do following tests.

Most tests have been done on the prototype implementation with 4 Seagate Cheetah Ultra320 (MODEL ST373307LC) disks. The machine running Iometer benchmark is a Xeon 2.4GHz PC server with an Agilent FC HBA card. We also evaluate the performance of an nSTOR disk array with FC interface disks under similar conditions for comparing. As shown in Figure 4, our prototype implementation is even better than nSTOR in most cases.

In order to evaluate the RAID algorithm performance on target board and compare the processing capacity, we have done a set of tests on both PC and IQ80321 platform. As shown in Figure 5(a), the platform A can do about 4 times more I/O per second than X. The source code is the same for the two platforms, which means we have not utilized AAU on IQ80321 board now.

The evaluation results in Figure 5(b) indicate the effects of AAU and zero-copy. We can see that the performance get considerable enhancement when we offload IOP321 by AAU doing XOR processing in RAID 5. The memory throughput is 617MB/s with AAU, it's much greater than computing XOR by IOP (only 77MB/s, with 64KB block size). The zero-copy feature also has much contribution to the enhancement by avoiding memory copy overhead.

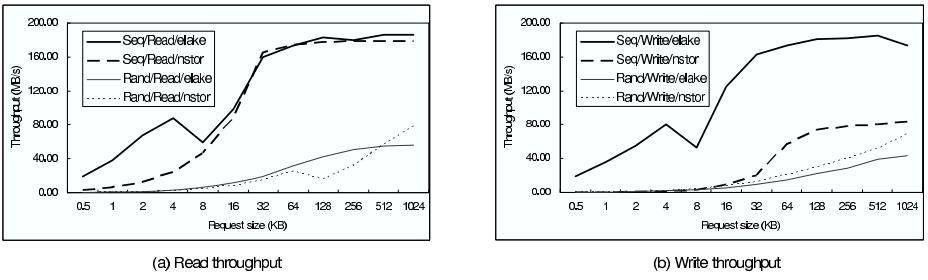


Fig. 4. This figure compares RAID 5 performance of our implementation (elake) with a medium level disk array from nSTOR. (Random or Sequential/Read or Write/elake or nstor). **(a)**Read throughput results; **(b)**Write throughput results

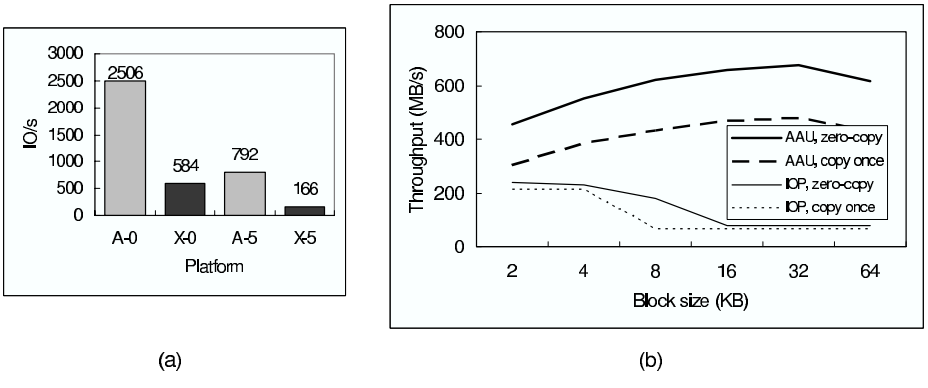


Fig. 5. (a) The test results compare processing capacity between general processors and XScale embedded processors. Each value is an average of data collected through about 1 hour. 'A' means PC platform with an AMD Athlon(tm) MP 2600+ CPU (2GHz) and 512MB memory; 'X' means IQ80321 board with IOP321 (600MHz) and 128MB memory. The 0 and 5 following platform sign represent RAID level 0 and 5 respectively. **(b)** The evaluation results compare effects on memory throughput by AAU engine and zero-copy policy with different block sizes

6 Related Works

RAID was declared at Berkeley in 1988 by D. A. Patterson, G. A. Gibson, and R. H. Katz in [1]. Challenging the well-known gap between I/O subsystem and processor, systems can alleviate the gap by distributing the controllers' and buses' loads across multiple, identical parts. RAIDs suffer from some limitations, such as bad performance for small writes, and limitation of scalability. There are considerable researches devoted to RAID-derived storage systems. Also some

new controller architectures have been developed, including AutoRAID [9] and TickerTAIP [10].

Several manufacturers provide hardware RAID array products, including LSI Logic, Mylex, Adaptec, Infortrend, Promise and 3ware, etc. [11] introduces managing issues of RAID on Linux systems.

7 Conclusions

RAID is today's standard solution for enterprise class storage systems. This project tries to develop a high performance RAID controller with our efforts. We have done these works:

- o Selected IOP321 as hardware platform and setup an embedded Linux operating system.
- o Designed RAID software and implemented a prototype system.
- o Evaluated prototype performance and did some comparisons between x86 and IOP321 platforms.

The performance evaluation seems incredible by approaching the full bandwidth of Fibre Channel connection, and we are planning to port the software to our customized IOP321 board for RAID controller in the near future.

8 Acknowledgments

This project is supported by the National Natural Science Foundation of China (No. 60273073) and National Key Project of Fundamental R & D of China (973 project, No. 2004CB318203). It is partially supported by Wuhan Elake Storage Technology Co., Ltd.

References

1. Patterson, D. A., Gibson, G. A., Katz, R. H.: A Case for Redundant Arrays of Inexpensive Disks (RAID). In Proceedings of the International Conference on Management of Data (SIGMOD) (June 1988)
2. Chen, P. M., Lee, E. K., Gibson, G. A., et al.: RAID: High-Performance, Reliable Secondary Storage. ACM Computing Surveys, Vol. **26**(2) (1994) 145–188
3. http://www.intel.com/design/IIO/ (December 2004)
4. Intel IQ80321 I/O Processor Evaluation Platform Board Manual. Document Number: 273521-006, November 7, 2002.
5. Yaghmour, K.: Building Embedded Linux Systems. O'Reilly & Associates (May 2003)
6. http://www.busybox.net/ (December 2004)
7. http://sources.redhat.com/redboot/ (December 2004)
8. http://www.iometer.org/ (December 2004)

9. Wilkes, J., Golding, R., Staelin, C., and Sullivan, T.: The HP AutoRAID Hierarchical Storage System. ACM Transactions on Computer Systems, Vol. **14**(1) (February 1996) 108–136
10. Cao, P., Lim, S. B., Venkataraman, S., and Wilkes J.: The TickerTAIP Parallel RAID Architecture. ACM Transactions on Computer Systems, Vol **12**(3) (August 1994)
11. Vadala, D.: Managin RAID on LINUX. O'Reilly & Associates (2003)

Component-Based Integration Towards a Frequency-Regulating Home Appliance Control System*

Weiqin Tong[1], Qinghui Luo[1], Zhijie Yin[1], Xiaoli Zhi[1], and Yuwei Zong[2]

[1] School of Computer Engineering and Science, Shanghai University, Shanghai, China
wqtong@mail.shu.edu.cn, luoqinghui@graduate.shu.edu.cn
[2] Shanghai Software Technology Development Center, Shanghai, China

Abstract. Conventional approaches to embedded software development are very costly, mainly because of their close reliance on application-dependant design, ad-hoc implementation, time-consuming performance tuning and verification. The resulting software is often hard to maintain, upgrade and customize. Component-based integration is an effective method to address the problem. As the result of our effort, an embedded software developing platform is constructed while developing an embedded control system for frequency-regulating home appliances. In this paper, the control system is described, and the developing platform is presented in detail, including an Application Component Library and a Fast Developing Tool. Component-based embedded software design and implementation are also put forward.

1 Introduction

There has been a great challenge, and a great opportunity as well, confronting the home appliances industry, namely to move toward the trend of home appliances informatization and frequency regulation. Informatization can attach new functions to home appliances, while frequency regulation technology can save 10%~30% of the energy used by home appliances.

This motivates an embedded control system for frequency-regulating home appliances. To reduce design time and development cost, and enable software customizing and tailoring, the reusable design concept is incorporated into the design of this system. Much attention is paid to one of the key technical problems and challenges—platforms and tools. An embedded system developing platform is built up, together with an Application Component Library and a Fast Developing Tool for frequency-regulating control software.

This paper is organized as follows. Section 2 gives an overview of the control system. Section 3 introduces the system developing platform and tool. Section 4 expatiates on the embedded software design and implementation based on software

* This work is supported in part by Science and Technology Commission of Shanghai Municipality under grant number 04dz15003.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 118-123, 2005.

component. Finally, section 5 identifies directions for further research and concludes this paper.

2 System Overview

The embedded control system consists of the following parts:

- ***Central control module.*** It is the kernel of the embedded control system, with the task of processing the information generated by other modules and coordinating their executions.
- ***Data acquisition and processing module.*** This module is in charge of collecting the status of the running system and environment, and doing preliminary analysis as well.
- ***Frequency regulating module.*** Frequency regulation technology is employed for carrying out the stepless speed variation to the alternating current motor. As the rotating speed of the alternating current motor is proportional to the frequency of electric current, the rotating speed of the motor can be adjusted by regulating the power frequency. This module implements the control algorithms.
- ***Frequency regulating algorithms remote update module.*** This module receives the program codes transferred through network interface and writes them into the FLASH ROM or EEPROM, in order to upgrade the driver routines, revise the software defects and achieve function extension.
- ***Communication module.*** This module enables the short distance communication with home appliances through serial port or infrared interface.
- ***Human-machine interface module.*** This module displays the running status of the home appliance on the LCD/LED installed on the control panel, and offers facility to control and adjust the appliance by touch screen or buttons on it, also provides means to gather the running information and control the appliance through remote controller.
- ***Remote monitoring and maintenance module.*** This module provides remote control and monitoring functions, and supports remote maintenance as well. Remote manipulator can control and monitor the appliance via a computer connected to the network, while maintenance staffs can do runtime maintenance and fault diagnosis according to the result by analyzing the running status that the appliance feeds back.
- ***Self-adjusting and individualization configuring module.*** The operating environments of home appliances may vary geographically and climatically; for instance, there are differences in temperature and humidity between the north and the south. This has great influence on the use and maintenance of home appliances. Self-adjusting module helps adjust appliances to ideal temperature and humidity according to different geographic and climatic conditions. Moreover, individualization configuring module keeps record of user's commands and preferences, and responds promptly later when the user issues a similar command or needs specific preferences.

The system is modeled based on software component, as an effort to support extensibility, and additional function modules can be incorporated easily.

3 System Developing Platform

The control system consists of hardware and software. A microcontroller based on ARM7TDMI™ is used for the target system. The embedded operating system Reworks [4] is adopted for our purpose. It supports hard real-time operation and is suitable for the control of electronic equipments, also provides an integrated development environment (IDE) named ReDe, supporting ARM processors with facilities like cross-compiling, linking, emulation debugging and downloading.

The OS running on the host machine is Windows 2000/NT/XP. The developing platform includes the following parts:

- ***Application Component Library***
- ***Fast Developing Tool for frequency regulating control software***
- ***Supporting tools like cross-compiling, linking, emulation debugging and downloading:*** Generate the binary code that can run on the target system and the debugging information.
- ***System design tools:*** Provide designers with a graphic interface for component-based software development, also interfaces and/or methods to invoke other tools.

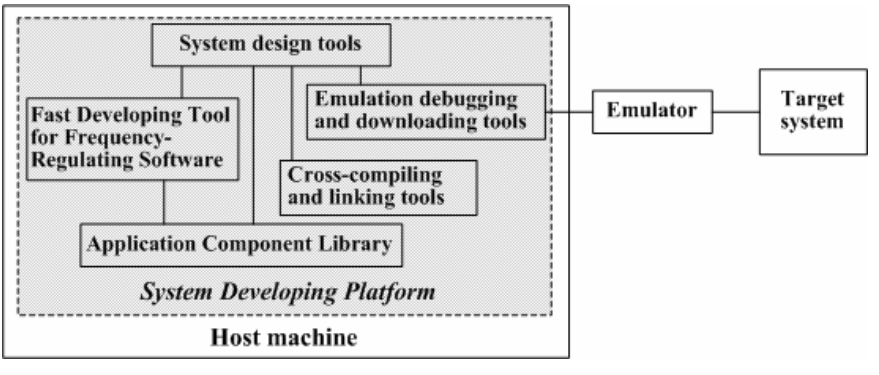


Fig. 1. Structure of the developing platform

3.1 Application Component Library

The component library is a repertory of components source codes and associated information, including component insertion or deletion, component search, components attributes table maintenance [1][2]. The component attributes include the types of target processors, locating and linking requirements, hardware requirements, related component information, etc. Some attributes can be adjusted according to specific operating environments.

To develop the software based on components, the software functionality is decomposed into different components. The principles and the methodologies of the decomposition, and the definitions of the components function field, may be different under specific situations [2]. In our system, necessary components are decomposed

out according to the principle of function independence, integrity, and functional cohesion. These components include but are not limited to:

- ***Data acquisition and processing component***
- ***Frequency regulating component***
- ***Communication component***
- ***Downloading component:*** writing code into FLASH ROM or EEPROM.
- ***Embedded operating system component:*** providing kernel modules of Reworks running on some typical ARM MCU, and the BOOTLOADER module.

Also, there are human-machine interface component and remote monitoring and maintenance component. These components can be decomposed into sub-components, if necessary.

3.2 Fast Developing Tool for Frequency-Regulating Control Software

The Fast Developing Tool (FDT) is based on the Application Component Library. It provides a Graphic User Interface (GUI) for users to configure typical static data and control parameters of the frequency regulating software. The Component Develop Module of FDT accesses the Application Component Library and fetches necessary components according to user's configuration. Furthermore, it unpacks the components to obtain the source code encapsulated in them. Then the Code Generating Module synthesizes static data and the source code of components to generate the program code of target software. Fig.2 describes the work model of FDT.

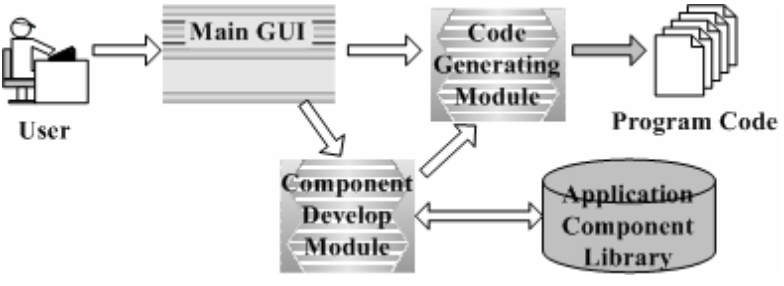


Fig. 2. FDT work model

4 Embedded Software Design Based on Component

Component-based software development means building an application system on the basis of the existing reusable components. When the Application Component Library is built up, or a component set of certain scale is collected, fast customization of the embedded software can be realized [3]. Now the developer's task is not to build a system from scratch, but to determine what components are needed according to the application system architecture, to adjust the components according to special requirements of the system, to add application-specific components where necessary,

and then to assemble these components into a complete system. Fig.3 shows the development flow chart.

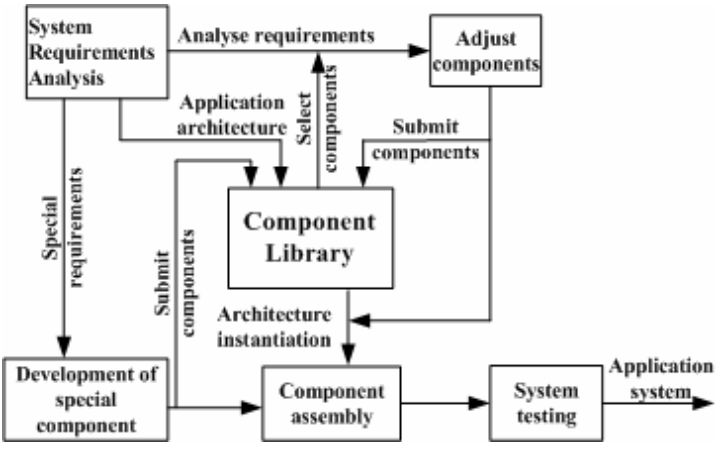


Fig. 3. Component-based software development flowchart

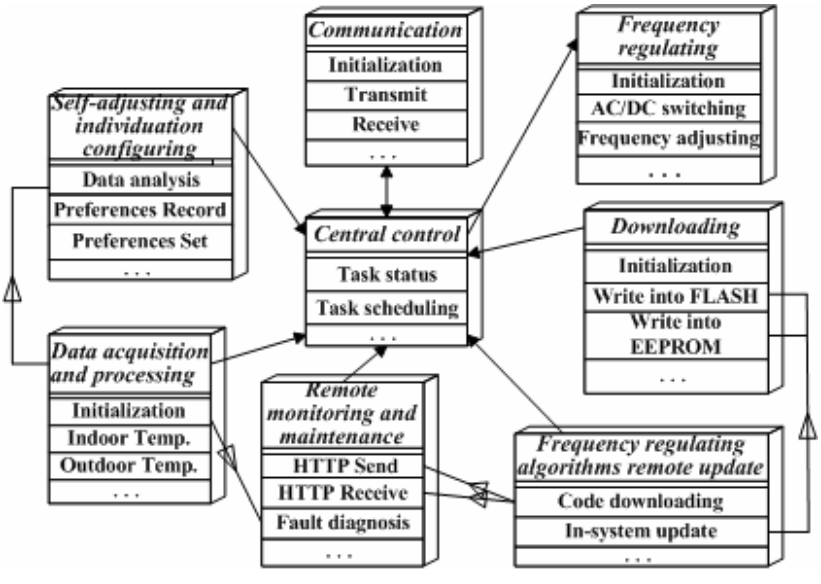


Fig. 4. Components and their relationships (The solid arrow represents relationship between central control component and other components, while the hollow arrow represents relationship among other components)

Components can be implemented either in the form of class in C++ or in the form of structure in C. Each component may consist of one or more classes (or structures). Functions of a component are implemented by different methods and interfaces provided by these classes. For example, `CDataAcquisition`, a class of the data acquisition and processing component, includes the following methods and interfaces:

```
void Init();
float getTempInside();
float getTempOutside();
float getHumidityInside();
```

The Init() method is responsible for the initialization of related hardware (I/O ports, for example), the other three "get" methods obtain the indoor temperature, the outdoor temperature and the indoor humidity, respectively.

Similarly, a structure named `Sdisplay` can be implemented to take over the role of LCD initialization and display in human-machine interface component. Its primary methods include:

```
void Init();
void setDisplayMode(TdisplayMode mode);
void D_printf(const char *format,...);
void clearScreen();
```

After the establishment of Application Component Library, what developers need to do is to outline the software architecture and choose the proper components to assemble a system. Fig.4 illustrates the cooperation relationships of the main components used in this system.

In practice, the relationships between components can be easily established by using some commercial Unified Modeling Language (UML) tools [5]. Hence the implementation part, i.e. coding, is greatly simplified.

5 Conclusions

A developing platform is established to assist the development of embedded control system for frequency-regulating electrical home appliances. Based on the platform, a prototype system is developed. Component-based approach is adopted to enhance software reusability and reduce system development time as different components can be developed and tested in parallel. Our future effort will be devoted to enrich the Application Component Library and improve the quality of components and the performance of our Fast Developing Tool as well.

References

1. Thomas Genßler, Oscar Nierstrasz, Bastiaan Schonhage, Components for Embedded Software [J]. CASE 2002 October 8-11, 2002, Grenoble, France.
2. Yang Fuqing, Mei Hong, Li Keqin, Software Reuse and Software Component Technology, ACTA ELECTRONICA SINICA, 27(2):68-75, Feb 1999.
3. M.Jenko, N.Medjeral, P.Butala, Component-based software as a framework for concurrent design of programs and platforms －an industrial kitchen appliance embedded system, Microprocessors and Microsystems 25 (2001) 287-296.
4. Reworks Reference Manual, East China Institute of Computing Technology, May 2004.
5. G. Booch, I. Jacobson, and J. Rumbaugh. Unified Modeling Language User Guide. Addison Wesley, 1997.

Design and Implementation of the System for Remote Voltage Harmonic Monitor

Kejin Bao [1, 2], Huanchuen Zhang [1], and Hao Shentu [2]

[1]Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[2]Jiangsu University, Zhenjiang 212013, China
bkj@ujs.edu.cn

Abstract. Take 8-bit microcontroller (C8051F005) as the nucleus and study a kind of voltage harmonic monitor device that satisfies the long-range monitor. The harmonic analysis algorithm adopts Fast Fourier Transform (FFT). This device has two work modes: "native" and "remote". It also can communicate with monitor center through telephone line, serial port, IC card, PDA (Personal Digital Assistant), etc.

Key Words: Harmonic test; remote-monitor; C8051F005 Microcontroller

1 Introduction

With the technical development of modern electronics and electrics, non-linear electric load in power grid increases considerably. This non-linear load causes aberration in the power grid and produces power grid harmonic. The power grid harmonic has become a social effect of pollution, and it is becoming more and more serious now. To eliminate the pollution, we must monitor and analyze the harmonic effectively in harmonic pollution area, then adopt valid measures. With the starting of power grid reforms, large number of unmanned transformer substations is rushing out now. Compared with the monitor device used in normal environment, the monitor device used in unmanned environment requires more reliability and remote communication capability. The wide usage of the monitor devices also makes cost an even more important issue. This paper introduces a new voltage harmonic monitor device that is based on an 8-bit microcontroller (C8051F005). This monitor device can monitor 1 - 31 power harmonic voltage in power lines in real time. It can record how long and how big voltage has been over the limit, and record the time of power going on and off. Both of the works are done in real time.

2 Hardware Constitution of Monitor Device

Real time volt and harmonic data are measured, analyzed and stored in monitor device. Over value time, power lost time and other information are calculated from previous data and are also stored in monitor device. These data can send back to monitor center by telephone line, serial port, IC card and PDA when they are needed.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 124-129, 2005.

In monitor center, these data are processed by certain software that runs on a personal computer and report documents can then be generated. The diagram of this system is shown in Fig.1.

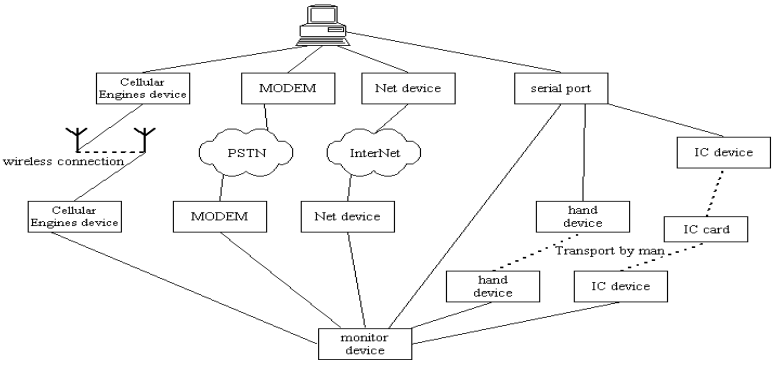


Fig. 1. Diagram of the system

Normal 8-bit MPU runs at a relative low speed which can not afford for harmonic process, so most of such monitor systems are based on 16-bit microcontrollers or are multi-CPU systems combined with DSP. But the C8051F005 produced by Cyganl Company, an 8-bit microcontroller, is fast enough to do such a work. It can run at a speed of 25MIPS which is fast enough for harmonic calculating. It is also integrated witch A/D adapter, RAM, FLASH ROM, voltage compare and a lot of I/O ports in the single chip and is suitable for unmanned environments.

The circuit architecture of Voltage Harmonic Monitor Device is shown in Fig.2.

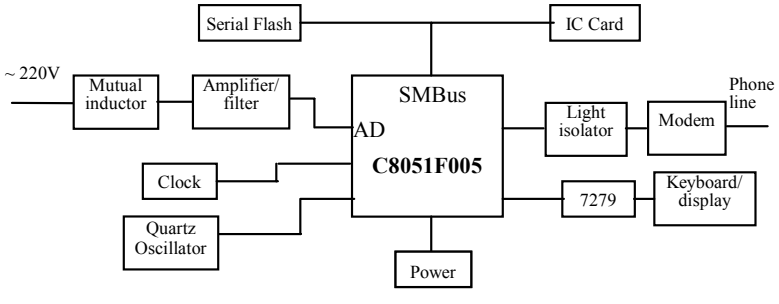


Fig. 2. The circuit architecture of Voltage Harmonic Monitor Device

3 Measuring Voltage Virtual Value and Voltage Harmonic

The key problem of voltage harmonic monitor device is that the virtual value of voltage and voltage harmonic must be accurately measured. From the architecture of the monitor device (Fig.2), it can be seen that measuring voltage need following steps: first, sample the 50Hz alternating currents voltage through mutual inductor; then

amplify and filter the sampled value and send the value into CPU; finally, the CPU samples and calculates virtual value of voltage and every power harmonic value.

3.1 Measuring the Virtual Value of Voltage

Measuring the virtual value of voltage adopts rectangle algorithm. Uniformly divide one period into N interval. Suppose *Vn* is instantaneous value on the n-th point. Because the interval is very short, we can assume that the voltage in this whole interval is *Vn*. Thus we can obtain the virtual value of voltage as follows:

$$V_{rms} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} V_n^2} \tag{1}$$

To increase the accuracy of voltage virtual value that we calculate, we sample the voltage many times within a second, and calculate the voltage virtual value according to every sampled data. All voltage virtual values we have calculated in one second are processed as follows: first, sort those values according to value size. After removing the maximum and the minimum ones, calculate the average value, and take this average as voltage virtual value in this second. Voltage harmonic monitor device refurbishes the voltage virtual value and displays it every second.

3.2 Measuring Voltage Harmonic

We calculate the harmonic component of alternating signals using DFT. Discrete Fourier transform and discrete inverse-Fourier transform are as follows:

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n) W^{nk} \qquad (0 \le K \le N-1) \tag{2}$$

$$x(n) = IDFT[X(K)] = \frac{1}{N}\sum_{k=0}^{N-1} X(k) W^{-nk} \quad (0 \le n \le N-1) \tag{3}$$

where: $W = e^{-j(\frac{2\pi}{N})}$.

Because imaginary number cannot appear in harmonic detecting, we can further adopt real-sequence FFT algorithm. It utilizes one FFT calculation to obtain two real-sequence FFT transforms, and reduces the computational complexity by about 1/2.

In this way, we can calculate $X(k)$ and $X(N-k)$ at the same time, then through conjugating $X(N-k)$ we also obtain $X^*(N-k)$. In other words, we make one DFT and obtain two DFTs of two sequences, because FFT simplifies the computational complexity by dividing one DFT into two sub-DFTs which can be created in one DFT transforming by using real FFT. After Fourier transforming of sampled data, we obtain corresponding 1 to 31 power harmonic virtual values. Every harmonic component is saved in two bytes, thus saving the result of one FFT needs at least 62 bytes of memory. According to the speed of CPU adopted by us, it can calculate 1-31

power harmonic 10 times per second. However, if we save all these harmonic calculation results in one second, the RAM in CPU cannot provide enough space. For simple reasons, the last calculated result is used as current result and is shown on 7-Segments LEDS. Because only total harmonic distortion (THD), total odd order harmonic distortion, total even order harmonic distortion are stored for statistic, only the max three THD data measured in one second is saved in memory，the third max THD is used as THD in this second. (Same as quality of electric energy supply standard of PRC). Both high respond speed and measure accuracy can be achieved by this method.

4 Improve the Precision of Measuring

Measuring accuracy is a very important issue in the design of 8-bit CPU based harmonic monitor device. To improve the measuring accuracy, we use 12-bit A/D for sampling voltage analog value. Every sampled data are expressed by two bytes. Moreover, several other new measures are also adopted for choosing sampling period, designing hardware and software filter wave.

4.1 Sample Time Determination

In order to reduce the leakage error in FFT transformation, the signal value should be sampled at exact time which divides the signal period by the same time span. This method is also called synchronic sampling. 20ms is used as a standard period in power grid, but it can not stay at the value. The normal implementation of synchronic method is to use PLL circuit. For the simplification of hardware, voltage compare, timer and software are used to measure the period time, compare the signal voltage value with the average volt value and record the time when the signal voltage is raising just above the average volt value. The time span between the record times is the signal period. Actual the compare voltage is not important. The period time should not be bigger or less than 5% of the standard period time; otherwise the result should be discarded, the average result of the period should be used for a better and more precise result.

The timer value is obtained by dividing the period of detected voltage by sample times. When the division is aliquant, there will be some errors. The maximum in these errors is smaller than the minimal timer resolution, and the maximal error value in the whole sampled sequence is smaller than the minimal value multiplying sampling times. As adopt C8051F005 (24 MHz) and 256 sampling points, the error is less than 12.8us. In order to further improve the sampling accuracy, the difference between the timer setting value and the actual value is recorded and accumulated. When the accumulated error exceeds minimal timer resolution, the sampling time is adjusted such that the maximal error value in the whole sampled sequences will be no more than the minimal timer resolution. For C8051F005 (24 MHz), the maximal error in one sampling sequence is only 0.05us.

4.2 Filtering the Sampling Signal

By using voltage mutual inductor, the input voltage can be reduced to the range suitable for sampling. In order to reduce disturbance of high frequency noise, active filter network is used to filter the voltage before it is sampled. The filter network is a low-pass network whose cutoff frequency must be lager than the concerned highest harmonic frequency. Its frequency response must be kept flat in the range where the frequency is less than the cutoff frequency; otherwise it may affect the result of harmonic measuring. The cutoff frequency of the active filter network is 1600Hz.

The signal passes an active filter hardware before sampling, the arithmetic mean of the voltage values measured in one second except the max and min value is calculated and is used as voltage value of this second. In practice, this voltage value is not very stable for Electro Magnetic interference, so we make a second filter: use the arithmetic mean of the values of this second and previous two seconds as current voltage value. After such process, the value is the same as the value measured by a voltage meter with accuracy of $\pm 0.5\%$ in the most of the time.

4.3 Minimize Quantization Noise

Over-sampling and averaging can be used to achieve higher ADC resolution measurements and SNR (signal-to-noise ratio) of analog-to-digital conversions. Over-sampling and averaging will improve the SNR and measurement resolution at the cost of increased CPU utilization and reduced throughput.

For each additional bit of resolution, the signal must be over-sampled by a factor of four.

$$f_{os} = 4^W f_s \tag{4}$$

where w is the number of additional bits of resolution desired, f_s is the original sampling frequency requirement, and f_{os} is the oversampling frequency.

Each doubling of the sampling frequency will lower the in-band noise by 3 dB, and increase the resolution of the measurement by 1/2 bit.

Use the f_{os} as sampling frequency, we accumulate (add 4^W consecutive samples together) then divide the total by 2^W (or right shift the total by W-bits). And this data is used as normal sample data. In practice, w=2 is used and obvious throughput reduction can be found.

5 Monitor Center Software Design

The software used on PC is programmed in VC++® 6.0 develop environment; PDF and CHM file are used as software help document. The html file of the CHM document is created by DREAMWARE® 3.0, and CHM document is generated by HTMLHELP® 1.3. PDF document is created by WORD® and ACROBAT®.

Report document is crated by WORD® and EXCEL®, so it can be opened on any PC with WORD® or EXCEL® installed on it. Because WORD® and EXCEL®

document is a kind of compound document, so it can not be created by simple use WINDOWS API such as ReadFile or WriteFile. The OFFICE® objects and VBA® (VB® Script of Application) function is used to generate these documents.

For simple reasons, ACCESS® is used as database management system. In consider of transportation reasons, only ODBC functions are used in the software. Special functions which can be run in ACCESS® only, such as fast data exchange in OFFICE software, are not used. So the software can be run under other DBMS with litter changes.

6 Conclusion

The voltage harmonic monitor device based on microcontroller C8051F005 can in real time monitor the 1 to 31 power harmonic voltage virtual values, total aberration rate, time and number of times of power cut, and the value of the voltage exceeding the limit. The device also has functions such as alarm, programming for detecting subsection time, and remote data transmitting.

This remote voltage harmonic monitor device has passed the test of Jiangsu Institute of Measurement &Testing Technology in May, 2003. This research has also passed the technology achievements identification from Department of Science and Technology of Jiangsu, China in Dec 28, 2003. Now, it has been in volume production. It provides low cost monitor instrument for power supply departments and let them master the state of power usage and analyze the quality of power grid.

Reference

1. Stefan Kunis, Daniel Potts, Fast spherical Fourier algorithms, Journal of Computational and Applied Mathematics 161 (2003) 75 – 98
2. Tian Xiaolin, Wang Jianhua, Liu Hongjun, Method of power harmonics analysis based on single chip processor and FPGA, Electrical Measurement & Instrumentation, 2004.2
3. Xiao Jian-hua, Wu Jin-pei, The Design of the Power Harmonics Detection System, Journal of Jishou University (Natural Science Edition), Vol . 21 No. 3, 2000.9
4. http://www.xhl.com.cn

Guaranteed Cost Control of Networked Control Systems: An LMI Approach

Shanbin Li, Zhi Wang, and Youxian Sun

National Laboratory of Industrial Control Technology
Institute of Modern Control Engineering
Zhejiang University, Hangzhou 310027, P.R.China
sbli@iipc.zju.edu.cn

Abstract. In this paper, networked control systems (NCS) is modeled as a discrete-time linear state-delayed system with norm-bounded uncertainty. Inspired by the so-called descriptor model transformation, a delay-dependent sufficient condition for the existence of a guaranteed cost controller for NCS is presented by a set of linear matrix inequalities (LMIs). The resulting controller can not only asymptotically stabilize the system but also guarantee an adequate level of performance. Theoretical analysis and illustrative results show that the control strategy presented in this paper is effective and feasible.

1 Introduction

Guaranteed cost control was firstly presented by Chang and Peng [1]. Its objective is to design a control system, which is not only stable but also guarantees an adequate level of performance. It is obviously a useful method in industrial control system. However, guaranteed cost control for networked control systems (NCS) wherein the control loops are closed through communication networks has not been studied extensively.

Up to now, the NCS has been an emerging research topic, which attracts increasing attention. The existence of networks in control systems brings about many problems including network-induced delays, jitter, packet losses as well as limited bandwidth [2]. It's well known that the network-induced delays introduced into the control loop can deteriorate the dynamic performance of systems. In worse case, these delays may be a main cause of potential system instability. Consequently, the basic focus in NCS involves the control of network-induced delays [3].

The network-induced delay issue of NCS is discussed using the guaranteed cost control approach in this paper. There are numerous stability results of guaranteed cost control for uncertain systems with time-delay, see e.g. [4]. In [4], a delay-dependent guaranteed cost control was developed for discrete-time state-delayed system by the so-called descriptor model transformation, which was firstly presented in [5]. Motivated by the transformation, this paper proposes a delay-dependent guaranteed cost control for NCS by modeling it into

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 130–136, 2005.

an uncertain discrete-time system with state-delay. The state feedback problem for NCS is formulated as a convex optimization over a set of LMIs, which can be very efficiently solved by interior-point methods [6]. At the same time, the optimal performance index sufficing that control law is given and the efficiency of guaranteed cost control strategy is demonstrated by an illustrative example.

2 Model Description

Assume that the uncertain discrete-time plant model is:

$$x(k+1) = (A + D\Delta(k)E_a)x(k) + (B + D\Delta(k)E_b)u(k), \tag{1}$$

where $x(k)$ is the state, $u(k)$ is the control input, A, B, D, E_a and E_b are real constant matrices. $\Delta(k)$ is an uncertain time-varying matrix stratifying the bound $\Delta^T(k)\Delta(k) \leq I$. All matrices are assumed to have compatible dimensions.

There are time-varying but bounded delays τ_k^{sc} and τ_k^{ca} in the control loops. Here τ_k^{sc} (τ_k^{ca}) is the communication delay between sensor and controller (controller and actuator) at time k respectively. If a static feedback controller is adopted, then τ_k^{sc} and τ_k^{ca} can be equivalently lumped together as a single delay $\tau(k) = \tau_k^{sc} + \tau_k^{ca}$. Furthermore, the bounds of network-induced delay can be obtained by the best and worst case analysis as [7]. Hence, the assumption $\underline{\tau} \leq \tau \leq \overline{\tau}$ is reasonable, where $\underline{\tau}$ and $\overline{\tau}$ are positive integers corresponding to minimum and maximum of $\tau(k)$. Then the controlled state information is given as:

$$u(k) = Kx(k - \tau(k)), \tag{2}$$

where K is the state-delayed feedback gain.

Given positive definite symmetric matrices Q_1 and Q_2, the cost function is considered as:

$$J = \sum_{k=0}^{\infty} \left[x^T(k)Q_1x(k) + u^T(k)Q_2u(k)\right]. \tag{3}$$

Associated with the cost function (3), the guaranteed cost controller (2) is defined as following:

Definition 1: *Consider the uncertain system (1) and the cost function (3). If there exists a control law $u(k)$ and a positive scalar J^* such that, for all admissible uncertainties, the closed-loop is stable and the closed-loop value of the cost function (3) satisfies $J \leq J^*$. Then J^* is said to be a guaranteed cost and $u(k)$ is said to be a guaranteed cost controller for the uncertain system (1).*

This paper aims to develop a controller as (2) which achieves value J^* as small as possible for uncertain systems in the networked setup.

3 Guaranteed Cost Control over Network

3.1 Analysis of Robust Performance

Applying a new Lyapunov-Krasovskii function, a new sufficient condition for the existence of the guaranteed cost controller for uncertain plant (1) and cost (5) in the networked setting will be established in this subsection.

Applying the controller (2) to plant (1), we will have the following system:

$$x(k+1) = A_1 x(k) + B_1 x(k - \tau(k)), \tag{4}$$

where $A_1 = A + D\Delta(k)E_a$, $B_1 = (B + D\Delta(k)E_b)K$. Associated with the system (4) is the cost function:

$$J = \sum_{k=0}^{\infty} x_e^T(k) Q x_e(k), \tag{5}$$

where $x_e^T(k) = [x^T(k),\ x^T(k - \tau(k))]$, $Q = diag\{Q_1,\ K^T Q_2 K\}$.

Theorem 1. *Consider the system (4) with the cost function (5) and time-varying but bounded delay $\tau(k) \in \left[\underline{\tau}, \overline{\tau}\right]$. The system (4) is asymptotically stable if there exists a scalar $\epsilon > 0$ and symmetric positive-definite matrices $P_1 \in \mathcal{R}^{n\times n}$, $R \in \mathcal{R}^{n\times n}$ and $S \in \mathcal{R}^{n\times n}$, matrices $P_2 \in \mathcal{R}^{n\times n}$, $P_3 \in \mathcal{R}^{n\times n}$, $W \in \mathcal{R}^{2n\times 2n}$ and $M \in \mathcal{R}^{2n\times n}$ such that the following matrix inequalities are satisfied:*

$$\Theta(\underline{\tau},\ \overline{\tau}) = \begin{bmatrix} \Gamma & P^T \begin{bmatrix} 0 \\ BK \end{bmatrix} - M & \begin{bmatrix} E_a^T \\ 0 \end{bmatrix} \\ * & -R + K^T Q_2 K & K^T E_b^T \\ * & * & -\epsilon I \end{bmatrix} < 0, \tag{6}$$

$$\begin{bmatrix} W & M \\ * & S \end{bmatrix} \geq 0, \tag{7}$$

where the $*$ represents block that is readily inferred by symmetry and:

$$\begin{aligned} \Gamma = {} & P^T \begin{bmatrix} 0 & I \\ A-I & -I \end{bmatrix} + \begin{bmatrix} 0 & I \\ A-I & -I \end{bmatrix}^T P + \epsilon P^T \begin{bmatrix} 0 & 0 \\ 0 & DD^T \end{bmatrix} P \\ & + \begin{bmatrix} \mu R + Q_1 & 0 \\ 0 & P_1 + \overline{\tau} S \end{bmatrix} + \overline{\tau} W + \begin{bmatrix} M & 0 \end{bmatrix} + \begin{bmatrix} M & 0 \end{bmatrix}^T, \end{aligned} \tag{8}$$

$$\mu = 1 + (\overline{\tau} - \underline{\tau}), \quad P = \begin{bmatrix} P_1 & 0 \\ P_2 & P_3 \end{bmatrix}. \tag{9}$$

Furthermore, the cost function (5) satisfies the following bound:

$$\begin{aligned} J \leq {} & x^T(0) P_1 x(0) + \sum_{l=-\overline{\tau}}^{-1} x^T(l) R x(l) + \sum_{\theta=-\overline{\tau}+1}^{0} \sum_{l=-1+\theta}^{-1} y^T(l) S y(l) \\ & + \sum_{\theta=-\overline{\tau}+2}^{-\underline{\tau}+1} \sum_{l=\theta-1}^{-1} x^T(l) R x(l), \end{aligned} \tag{10}$$

where $y(l) = x(l+1) - x(l)$.

Proof. Because of the limit of pages, the detail proof procedure is omitted here. Q.E.D.

3.2 Controller Design

A parameterized representation of the guaranteed cost control laws in terms of the feasible solutions to a set of LMIs will be presented in this subsection.

It's noted that the upper bound (10) depends on the initial condition of system (4), which will bring some difficulties in the solution to Theorem 1. In order to remove the dependence on the initial condition, we suppose that the initial state of system (4) is arbitrary but belongs to the set $\mathcal{S} = \{x(l) \in \mathcal{R}^n : \ x(l) = Uv_i, \ v_i^T v_i \leq 1, \ l = -\overline{\tau}, -\overline{\tau}+1, \ \ldots, \ 1, \ 0\}$, where U is a given matrix. Then the cost bound (10) leads to:

$$J \leq \lambda_{\max}(U^T P_1 U) + \rho_1 \lambda_{\max}(U^T R U) + \rho_2 \lambda_{\max}(U^T S U). \tag{11}$$

where $\lambda_{\max}(\cdot)$ denotes the maximum eigenvalue of matrix $(\cdot)$, $\rho_1 = \mu(\overline{\tau} + \underline{\tau})/2$ and $\rho_2 = 2\overline{\tau}(\overline{\tau}+1)$. Based on (11), the controller design theorem is given as following:

Theorem 2. *Consider the system (4) with the cost function (5) and time-varying but bounded delay $\tau(k) \in \left[\underline{\tau}, \overline{\tau}\right]$. Suppose that for a prescribed scalar δ, there exists a state feedback gain K such that the control law (2) with state-delay is a guaranteed cost controller if there exists a scalar $\epsilon > 0$, matrices $X > 0, \ Y, \ Z, \ F, \ L > 0, \ \bar{S} > 0, \ \bar{W}_1, \ \bar{W}_2, \ \bar{W}_3$, such that the following matrix inequalities are satisfied:*

$$\begin{bmatrix} \Psi_1 & \Psi_2 & 0 & XE_a^T & \overline{\tau}Z^T & 0 & X & Z^T \\ * & \Psi_3 & (1-\delta)BF & 0 & \overline{\tau}Y^T & 0 & 0 & Y^T \\ * & * & -L & F^T E_b^T & 0 & F^T & 0 & 0 \\ * & * & * & -\epsilon I & 0 & 0 & 0 & 0 \\ * & * & * & * & -\overline{\tau}\bar{S} & 0 & 0 & 0 \\ * & * & * & * & * & -Q_2^{-1} & 0 & 0 \\ * & * & * & * & * & * & -Q_1^{-1} & 0 \\ * & * & * & * & * & * & * & -X \end{bmatrix} < 0, \tag{12}$$

$$\begin{bmatrix} \bar{W}_1 & \bar{W}_2 & 0 \\ * & \bar{W}_3 & \delta BF \\ * & * & X\bar{S}^{-1}X \end{bmatrix} \geq 0, \tag{13}$$

where

$$\Psi_1 = Z + Z^T + \mu L + \overline{\tau}\bar{W}_1, \tag{14}$$

$$\Psi_2 = Y + X(A-I)^T - Z^T + \overline{\tau}\bar{W}_2 + \delta F^T B^T, \tag{15}$$

$$\Psi_3 = -Y - Y^T + \overline{\tau}\bar{W}_3 + \epsilon DD^T. \tag{16}$$

Furthermore, a guaranteed cost control law is given by (2) with $K = FX^{-1}$ and the corresponding cost function satisfies:

$$J \leq \lambda_{\max}(U^T X^{-1} U) + \rho_1 \lambda_{\max}(U^T X^{-1} L X^{-1} U) + \rho_2 \lambda_{\max}(U^T \bar{S}^{-1} U). \tag{17}$$

Proof. By Sherman-Morrison matrix inversion formula, we have:

$$P^{-1} = \begin{bmatrix} P_1^{-1} & 0 \\ -P_3^{-1}P_2P_1^{-1} & P_3^{-1} \end{bmatrix}. \tag{18}$$

Let $X = P_1^{-1}$, $Y = P_3^{-1}$ and $Z = -P_3^{-1}P_2P_1^{-1}$. Similar to [4], we have to restrict M to $\delta P^T \begin{bmatrix} 0 \\ BK \end{bmatrix}$ in order to obtain an LMI, where δ is a scalar parameter determined by designer.

Pre- and post-multiplying $diag\{P^{-1}, P_1^{-1}, I\}$ and its transpose to (6), respectively. Pre- and post-multiplying $diag\{P^{-1}, P_1^{-1}\}$ and its transpose to (7), respectively. We further denote $L = P_1^{-1}RP_1^{-1}$, $F = KP_1^{-1}$, $\bar{S} = S^{-1}$ and $(P^{-1})^T W P^{-1} = \begin{bmatrix} \bar{W}_1 & \bar{W}_2 \\ * & \bar{W}_3 \end{bmatrix}$. Applying the Schur complement and by expansion of the block matrices, the theorem is proved. Q.E.D.

From (17), we establish the following inequalities:

$$\begin{bmatrix} -\alpha I & U^T \\ * & -X \end{bmatrix} < 0, \quad \begin{bmatrix} -\beta I & U^T \\ * & -XL^{-1}X \end{bmatrix} < 0, \quad \begin{bmatrix} -\gamma I & U^T \\ * & -\bar{S} \end{bmatrix} < 0, \tag{19}$$

where α, β and γ are scalars to be determined. However, it is noted that the condition (13) and (19) are no more LMI conditions because of the terms $XL^{-1}X$ and $X\bar{S}^{-1}X$, respectively. Note that for any matrix $X > 0$, we have $X\bar{S}^{-1}X \geq 2X - \bar{S}, XL^{-1}X \geq 2X - L$.

Given a prescribed scalar δ, $\underline{\tau}$ and $\overline{\tau}$, the design problem of the optimal guaranteed cost controller can be formulated as the following LMI eigenvalue problem:

$$\textbf{OP:} \min_{\epsilon, X, Y, Z, F, L, \bar{S}, \bar{W}_1, \bar{W}_2, \bar{W}_3} (\alpha + \rho_1\beta + \rho_2\gamma)$$

$$\text{s.t.} \begin{cases} \text{(i)} \quad \text{Equation(12)}, \\ \text{(ii)} \quad \begin{bmatrix} \bar{W}_1 & \bar{W}_2 & 0 \\ * & \bar{W}_3 & \delta BF \\ * & * & 2X - \bar{S} \end{bmatrix} \geq 0, \\ \text{(iii)} \quad \begin{bmatrix} -\alpha I & U^T \\ * & -X \end{bmatrix} < 0, \ \begin{bmatrix} -\beta I & U^T \\ * & -2X + L \end{bmatrix} < 0, \ \begin{bmatrix} -\gamma I & U^T \\ * & -\bar{S} \end{bmatrix} < 0. \end{cases} \tag{20}$$

It is clear that the above optimization problem (20) is a convex optimization problem and can be effectively solved by existing LMI software [8]. Using this solution, we can calculate the upper bound of (11). This bound will be considered as a function of δ, $\underline{\tau}$ and $\overline{\tau}$. We select its minimum as a suboptimal value for the corresponding upper bound of the cost function (10).

4 Illustrative Example

In this section, a numerical example is presented to illustrate how to solve the optimization problem proposed in this paper and realize the networked control law. The plant is given as follows:

$$A = \begin{bmatrix} 1.13 & 0 \\ 0.16 & 0.478 \end{bmatrix}, \; B = \begin{bmatrix} 0.2 & 0.1 \\ 0 & -0.1 \end{bmatrix}, \; D = \begin{bmatrix} 0 & 0 \\ 0.1 & 0 \end{bmatrix}, \\ E_a = \begin{bmatrix} 0.1 & 0 \\ 0.1 & -0.1 \end{bmatrix}, \; E_b = \begin{bmatrix} 0 & 0.1 \\ 0.1 & 0 \end{bmatrix}, \tag{21}$$

and the simulation parameters are given as: $Q_1 = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$, $Q_2 = 0.1$, $U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. The eigenvalues of A are obtained as $\lambda_1 = 0.478$ and $\lambda_2 = 1.13$, which means the plant (21) is open-loop unstable.

We first choose $\delta = 1$ and $\underline{\tau} = 1$, $\overline{\tau} = 3$. By Theorem 2 and using Matlab command `mincx` of LMI-toolbox [8], we have $J_{cost} = 1.1357 \times 10^3$, $K = \begin{bmatrix} -1.1303 & -0.2577 \\ -0.3535 & -0.0897 \end{bmatrix}$. On the premise of assuring the feasibility to optimization problem (20), we increase δ from 0.7 to 1 by step 0.01, and $\overline{\tau}$ from $\underline{\tau}$ to 4 by step 0.1. Then a three-dimensional graph of cost, δ and $\overline{\tau}$ is obtained as Figure 1, where $J(1,1)$ is the cost obtained when $\delta = 0.96$, $\underline{\tau} = \overline{\tau} = 1$. As shown, increasing $\overline{\tau}$ results in an increase of cost, namely deteriorates the system performance. This validates the aforementioned analysis.

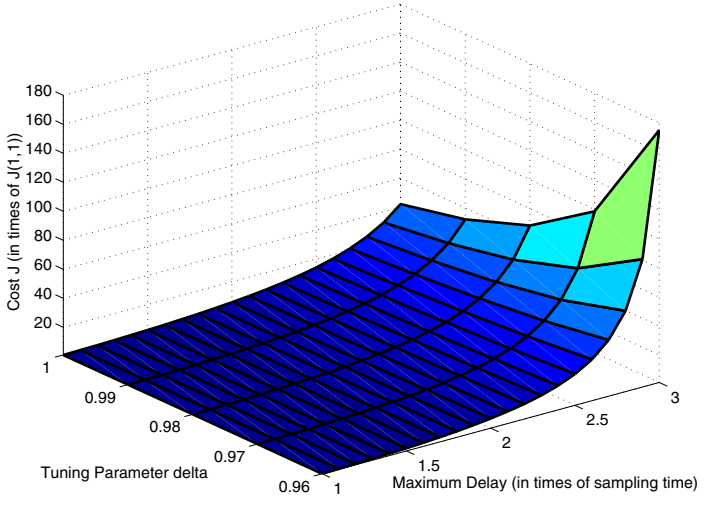


Fig. 1. The cost as a function of δ and delay $\tau \in [\underline{\tau}, \; \overline{\tau}]$

5 Conclusions

Network-induced delays in the networked control system are inevitable and have a significant effect on system performance. This paper addresses networked control systems (NCS) within the framework of a discrete-time linear state-delayed system with norm-bounded uncertainty. Based on the model, a delay-dependent sufficient condition for the existence of a guaranteed cost controller for NCS is presented by a set of linear matrix inequalities (LMIs). The resulting controller can not only asymptotically stabilize the system but also guarantee an adequate level of performance. It should be noted that the proposed results can be applicable to NCS, wherein the network-induced delay is random and bounded. It's also noted that the results can be extended to the case where the delay is not only shorter than one sampling time but also longer than one sampling time.

References

1. Chang, S., Peng, T.: Adaptive guaranteed cost control of systems with uncertain parameters. IEEE Trans. On Automat. Contr. **17** (1972) 474–483
2. Tipsuwan, Y., Chow, M.Y.: Control Methodologies in Networked Control Systems. Control Engineering Practice **11** (2003) 1099–1111
3. Krtolica, R., Özgüner, Ü., Chan, H., Göktas, H., Winkelman, J., Liubakka, M.: Stability of linear feedback systems with random communication delays. Int. J. Control **59** (1994) 925–953
4. Chen, W.H., Guan, Z.H., Lu, X.: Delay-dependent guaranteed cost control for uncertain discrete-time systems with delay. IEE Proc.-Control Theory Appl. **150** (2003) 412–416
5. Fridman, E., Shaked, U.: A descriptor system approach to h_∞ control of linear time-delay systems. IEEE Trans. Automat. Contr. **47** (2002) 253–270
6. Boyd, S., Ghaoui, L.E., Balakrishnan, E.F.V.: Linear Matrix Inequalities in System and Control Theory. SIAM, Philadelphia (1995)
7. Castelpietra, P., Song, Y.Q., Francoise, S.L., Attia, M.: Analysis and simulation methods for performance evaluation of a multiple networked embedded architecture. IEEE Transactions on Industrial Electronics **49** (2002) 1251–1264
8. Ghainet, P., Nemirovski, A., Laub, A.J., Chilali, M.: LMI Control Toolbox-for Use with Matlab. The Math Works Inc. (1995)

Robust Tuning of Embedded Intelligent PID Controller for Induction Motor Using Bacterial Foraging Based Optimization

Dong Hwa Kim

Department of Instrumentation and Control Eng., Hanbat National University,
16-1 San Duckmyong-Dong Yuseong-Gu, Daejon City, Korea, 305-719.
kimdh@hanbat.ac.kr

Abstract. In this paper, design approach of PID controller with rejection function against external disturbance in motor control system is proposed using bacterial foraging based optimal algorithm. To design disturbance rejection tuning, disturbance rejection conditions based on H_∞ are illustrated and the performance of response based on the bacterial foraging is computed for the designed PID controller as ITSE (Integral of time weighted squared error). Hence, parameters of PID controller are selected by bacterial foraging based optimal algorithm to obtain the required response.

1 Introduction

A Proportional – Integral – Derivative (PID) controller has been widely used in the most industrial processes despite continual advances in control theory. Most of the PID tuning rules developed in the past years use the conventional method such as frequency-response methods [1]. This method needs a highly technical experience to apply since they provide simple tuning formulae to determine the PID controller parameters. In case of the Ziegler-Nichols rule tuning technique, it often leads to a rather oscillatory response to set-point changes. Despite the fact that many PID tuning methods are available for achieving the specified GPM, they can be divided into two categories. On the other hand, since natural selection of bacterial foraging tends to eliminate animals with poor foraging strategies for locating, handling, and ingesting food, optimization models can be provided for social foraging where groups of parameters communicate to cooperatively forage in engineering. In this paper, an intelligent tuning method of PID controller by bacterial foraging based optimal algorithm is suggested for robust control with disturbance rejection function on control system of motor control loop.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 137-142, 2005.

2 PID Controller Tuning with Disturbance Rejection Function

2.1 Condition for Disturbance Rejection

In Fig. 1, the disturbance rejection constraint can be given by [7,8]

$$\max_{d(t)\in D}\frac{\|Y\|}{\|d\|}=\left\|\frac{w(s)}{1+K(s,c)G(s)}\right\|_{\infty}\langle\delta\,. \tag{1}$$

Here, $\delta\langle 1$ is constant defining by the desired rejection level and $\|\bullet\|_{\infty}$ denotes the H_{∞}-norm, which is defined as $\|G(s)\|_{\infty}=\max_{\omega\in[0,\infty)}|G(j\omega)|$.

The disturbance rejection constraint becomes

$$\left\|\frac{w(s)}{1+K(s,c)G(s)}\right\|_{\infty}=$$

$$\max_{\omega\in[0.\infty)}\left(\frac{w(j\omega)w(-j\omega)}{1+K(j\omega,c)G(j\omega,c)K(-j\omega,c)G(-j\omega,c)}\right)^{0.5}$$

$$=\max_{\omega\in[0.\infty)}\left(\sigma(\omega,c)\right)^{0.5} \tag{2}$$

The controller K (s, c) is written as $K(s,\,c)=c_1+\frac{c_2}{s}+c_3 s$ and the vector c of the controller parameter is given by $c=\left[c_1,\,c_2,\,c_3\right]^T$. Hence, the condition for disturbance rejection is given as $\max_{\omega\in[0.\infty)}\left(\sigma(\omega,c)\right)^{0.5}\langle\delta$.

2.2 Performance Index for Disturbance Rejection Controller Design

The performance index defined as ITSE (Integral of the Time-Weighted Square of the Error) is written by $PI=\int_0^{\infty}t(E(t))^2dt$, $E(s)=\frac{B(s)}{A(s)}=\frac{\sum_{j=0}^{m}b_j s^{m-1}}{\sum_{i=0}^{n}a_i s^{n-1}}$. (3)

Because *E(s)* contains the parameters of the controller *(c)* and plant, the value of performance index, *PI* for a system of n*th* order can be minimized by adjusting the vector c as $\min_{c} PI(c)$. The optimal tuning proposed in this paper is to find the vector c, so that the ITSE performance index *PI (c)* is a minimum using bacterial algorithm and the constraint $\max_{\omega\in[0.\infty)}\left(\sigma(\omega,c)\right)^{0.5}\langle\delta$ is satisfied through real coded bacterial algorithms.

3 Behavior Characteristics and Modeling of Bacteria Foraging

3.1 Overview of Chemotactic Behavior of E. coli.

This paper considers the foraging behavior of E. coli, which is a common type of bacteria as in reference 4-5. Its behavior to move comes from a set of up to six rigid 100–200 rps spinning flagella, each driven as a biological motor. An E. coli bacterium alternates between running and tumbling. Running speed is 10–20 $\mu m/\sec$, but they cannot swim straight. Mutations in E. coli affect the reproductive efficiency at different temperatures, and occur at a rate of about 10^{-7} per gene and per generation. E. coli occasionally engages in a conjugation that affects the characteristics of a population of bacteria. Since there are many types of taxes that are used by bacteria such as aerotaxis (it are attracted to oxygen), light (phototaxis), temperature (thermotaxis) and magnetotaxis, it can be affected by magnetic lines of flux. Some bacteria can change their shape and number of flagella which is based on the medium to reconfigure in order to ensure efficient foraging in a variety of media.

3.2 Optimization Function of Bacterial Swarm Foraging

The main goal based on bacterial foraging is to apply in order to find the minimum of $P(\phi)$, $\phi \in R^n$, not in the gradient $\nabla P(\phi)$. Here, when ϕ is the position of a bacterium, and $J(\phi)$ is an attractant-repellant profile. A neutral medium, and the presence of noxious substances, respectively can be shown by

$$H(j,k,l) = \{\phi^i(j,k,l) \mid i = 1,2,...,N\}. \tag{4}$$

Equation represents the positions of each member in the population of the N bacteria at the jth chemotactic step, kth reproduction step, and lth elimination-dispersal event. Let $P(i, j, k, l)$ denote the cost at the location of the ith bacterium $\phi^i(j,k,l) \in R^n$. Reference [20, 21] let

$$\phi^i = (j+1,k,l) = \phi^i(j,k,l) + C((i)\varphi(j), \tag{5}$$

so that $C(i)>0$ is the size of the step taken in the random direction specified by the tumble. If at $\phi^i(j+1,k,l)$ the cost $J(i, j+1, k, l)$ is better (lower) than at $\phi^i(j,k,l)$, then another chemotactic step of size $C(i)$ in this same direction will be taken and repeated up to a maximum number of steps N_s. N_s is the length of the lifetime of the bacteria measured by the number of chemotactic steps. Functions $P_c^i(\phi)$, i=1, 2, . . . , S, to model the cell-to-cell signaling via an attractant and a repellant is represented by

$$\begin{aligned} P_c(\phi) = \sum_{i=1}^{N} P_{cc}^i = \sum_{i=1}^{N} \left[-L_{attract} \exp\left(-\delta_{attract} \sum_{j=1}^{n} (\phi_j - \phi_j^i)^2 \right) \right] \\ + \sum_{i=1}^{N} \left[-K_{repellant} \exp\left(-\delta_{attract} \sum_{j=1}^{n} (\phi_j - \phi_j^i)^2 \right) \right], \end{aligned} \tag{6}$$

When we where $\phi = [\phi_{1,...,}\phi_p]^T$ is a point on the optimization domain, L_{attract} is the depth of the attractant released by the cell and $\delta_{attract}$ is a measure of the width of the attractant signal. $K_{repellant} = L_{attract}$ is the height of the repellant effect magnitude), and $\delta_{attract}$ is a measure of the width of the repellant. The expression of $P_c(\phi)$ means that its value does not depend on the nutrient concentration at position ϕ. Model use the function $P_{ar}(\phi)$ to represent the environment-dependent cell-to-cell signaling as $P_{ar}(\phi) = \exp(T - P(\phi))P_c(\phi)$, where T is a tunable parameter. Model considers minimization of $P(i, j, k, l) + P_{ar}\left(\phi^i(j,k,l)\right)$, so that the cells will try to find nutrients, avoid noxious substances, and at the same time try to move toward other cells, but not too close to them. The function $P_{ar}\left(\phi^i(j,k,l)\right)$ implies that, with M being constant, the smaller $P(\phi)$, the larger $P_{ar}(\phi)$ and thus the stronger attraction, which is intuitively reasonable. In tuning the parameter M, it is normally found that, when M is very large, $P_{ar}(\phi)$ is much larger than $J(\phi)$, and thus the profile of the search space is dominated by the chemical attractant secreted by E. coli. This paper describes the method in the form of an algorithm to search optimal value of PID parameter.

[step 1] Initialize parameters n, N, N_C, N_S, N_{re}, N_{ed}, P_{ed}, $C(i)$ $(i=1,2,...,N)$, ϕ^i, and random values of PID parameter. Where, n: Dimension of the search space (Each Parameter of PID controller), N: The number of bacteria in the population, N_C : chemotactic steps, N_{re} : The number of reproduction steps, N_{ed} : the number of elimination-dispersal events, P_{ed} : elimination-dispersal with probability, C(i): the size of the step taken in the random direction specified by the tumble. The controller parameter is searched in the range of Kp=[0 30], Ti=[0 30], and Td=[0 30].

[step 2] Elimination-dispersal loop: $l=l+1$

[step 3] Reproduction loop: $k=k+1$

[step 4]Chemotaxis loop: $j=j+1$

[step 5] If $j < N_C$, go to step 3. In this case, continue chemotaxis, since the life of the bacteria is not over.

[step 6] Reproduction:

[step 7] If $k < N_{re}$, go to [step 3]. In this case, we have not reached the number of specified reproduction steps, so we start the next generation in the chemotactic loop.

[step 8] Elimination-dispersal: For $i = 1,2...,N$, with probability P_{ed}, eliminate and disperse each bacterium. To do this, if you eliminate a bacterium, simply disperse one to a random location on the optimization domain. If $l < N_{ed}$, then go to [step 2]; otherwise end.

4 Simulations and Discussions

Fig. 1 shows the step response to variation of chemotactic size. When step size is 0.15, response is best response. Fig. 2 is comparison of results by GA (genetic algorithm), immune algorithm, and bacterial foraging. Fig. 3 is representing search process of

performance index (ITSE) by bacteria foraging and Fig. 4 is search process to have optimal PID parameters by bacteria foraging.

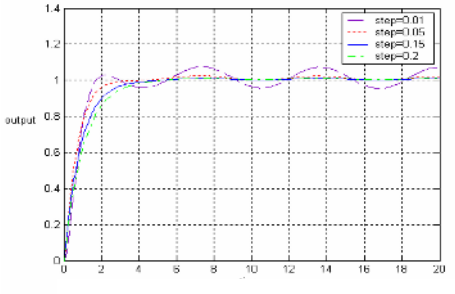


Fig.1. Step response by variation of chemotactic step size

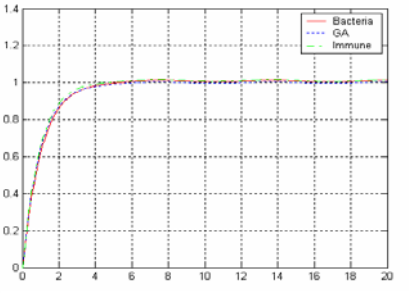


Fig.2. Comparison of each optimal algorithm. (GA, Immune algorithm, Bacteria Foraging)

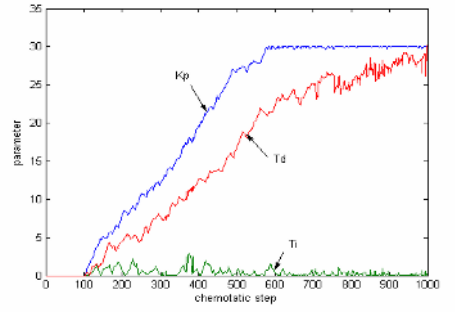


Fig.3. Search process of performance index (ITSE) by Bacteria Foraging

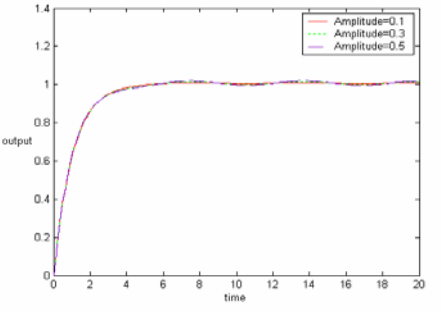


Fig.4. Search process of optimal PID parameters by Bacteria Foraging

5 Conclusions

Up to now, the PID controller has been used to operate the process loops including motor control. However, achieving an optimal PID gain is very difficult for the control loop with disturbances. Since natural selection of animal tends to eliminate animals with poor foraging strategies for locating, handling, and ingesting food, they obtain enough food to enable them to reproduce after many generations, poor foraging strategies are either eliminated or shaped into good ones redesigned. Therefore, optimization approach can be provided for social foraging where groups of parameters communicate to cooperatively forage in engineering.

Table 1. Comparison of PID parameter and ITSE of each optimal algorithm.

	Bacteria Foraging	GA[1]	Immune Algorithm
Kp	29.901	29.992	29.739
Ti	0.25813	0.0001	0.39477
Td	30	28.3819	27.277
ITSE	0.000668	0.000668	0.0006352

In this paper, an intelligent tuning method of PID controller by bacterial foraging based optimal algorithm is suggested for robust control with disturbance rejection function on control system of motor control loop. Simulation results are showing satisfactory responses. The object function can be minimized by gain selection for control, and the variety gain is obtained as shown in Table 1. The suggested controller can also be used effectively in the control system as seen from Figs. 1-4.

References

1. 2. J. X. Xu, C. Liu, and C. C. Hang: Tuning of Fuzzy PI Controllers Based on Gain/Phase Margin Specifications and ITAE Index. ISA Transactions 35 (1996) 79-91.
2. Dong Hwa Kim: Intelligent tuning of a PID controller with robust disturbance rejection function using an immune algorithm. Proc. Int. Conf. Knowledge-based intelligent information and engineering systems. Springer-Verlag (2004) 57-63.
3. PASSINO, K. M.: Biomimicry of Bacterial Foraging for Distributed Optimization and Control. IEEE Control Systems Magazine (2002)
4. Ching-Hung Lee, Yi Hsiung Lee, and Ching Ch-eng Teng: A novel robust PID controllers design by Fuzzy Neural network. Proceedings of the American Control Conference, Anchorage, May 8-10, (2002) 1561-1566
5. Dong Hwa Kim: Robust PID controller tuning using multiobjective optimization based on clonal selection of immune algorithm. Proc. Int. Conf. Knowledge-based intelligent information and engineering systems. Springer-Verlag (2004) 50-56.

The Customizable Embedded System for Seriate Intelligent Sewing Equipment

Kailong Zhang, Xingshe Zhou, Ke Liang, and Jianjun Li

School of Computer, Northwestern Polytechnical University Xi'an, Shaanxi, 710072, China
{kl.zhang, zhouxs}@nwpu.edu.cn

Abstract: Today，the development of sewing technology has shown the seriate intelligent trend, and one important factor is the embedded system technology. After careful research and analysis, this paper brings forward a customizable embedded system architecture, which is made up of customizable embedded hardware platform, customizable embedded OS and component-based embedded software. We have used this architecture to design new electro-pattern sewing machine successfully.

1 Introduction

With the fast development of embedded system technique, the industry of sewing equipment has entered an intelligent era after a long mechanical and electric sewing period. The new generation of electro-sewing equipments, which adopts the advanced intelligent embedded system, can do more complex and accurate sewing tasks, and is easier to operate.

Having analyzed the features of intelligent sewing machines, this paper presents a new customizable embedded system for the seriate intelligent requirement, which includes a customizable hardware platform, a customizable embedded software platform, a sewing domain-oriented embedded component set, development methods and so on.

2 Customizable Embedded Hardware Platform

An intelligent sewing machine is mainly made up of mechanical-electro devices and one embedded system. In such a machine, the embedded system makes all the electro devices act coordinately and can interact with user.

Between different sewing machines, there are many differences, such as the type of mechanical units, electric units and the control system. In order to explain this question, we compare the electro-lockstitch sewing machine with the electro-pattern sewing machine here. First, they are designed for different purposes. The former is mainly to execute two-dimensional linear sewing tasks, while the latter is for three-dimensional pattern sewing application. Second, they are different in mechanism. An electro-lockstitch sewing machine needs a main-shaft motor and some mechanical

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 143-149, 2005.

connecting rods to control the needle and move fabric backwards and forwards, while an electro-pattern one requires two step motors and one servomotor to achieve coordinated movement[1][2]. Of course, except these differences, there are some similarities among different ones in many aspects. From the simpler electro-lockstitch sewing machine to the more complex computerized embroidery machine, they all adopt the manipulative mode of "embedded system + electric units", and consist of motor driver, auto-threadtrimming set and other apparatuses controlled by embedded system. So, we can make the following conclusion primarily: the differences between them are only the type, number and manipulative approaches of motors and pneumatic equipments.

On the foundation, we put forward a new opening hardware platform that includes the foundational platform and special devices in this paper. The foundational platform includes general equipments and interfaces required by the seriate sewing machines, such as embedded processor, basic I/O, USB, Ethernet interface, etc. The special devices are required only by given intelligent equipment, for instance the motor controller, pneumatic units, data acquisition devices, and what not. With such opening architecture, embedded hardware system can be configured easily. And the nice extensibility of such architecture can also guarantee the developer to produce new intelligent sewing machines. The customizable hardware architecture for seriate sewing machines is shown in Fig1.

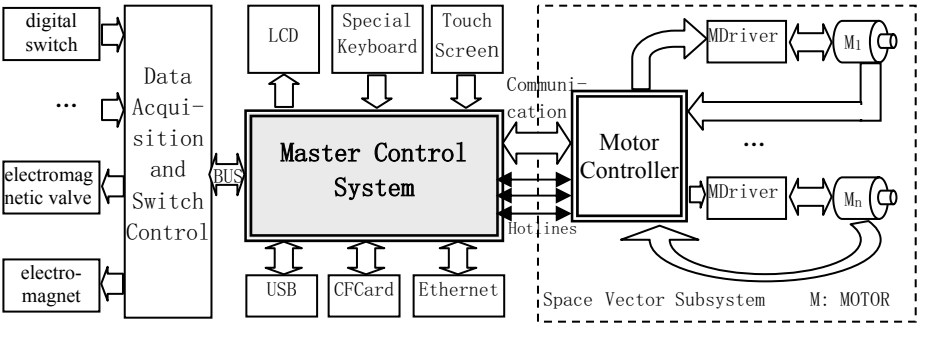


Fig. 1. Opening Architecture of Embedded Hardware for Seriate Intelligent Sewing Equipment

Therefore, the hardware platform of new sewing equipment can be developed conveniently by customizing the number of switch, motor and motor driver, and the interfaces of communication and control. For example, the typical hardware configuration for electro-pattern sewing machine can be the following: foundational hardware platform, touching-screen, LCD, CF card, 2-way digital switches, 3-way data acquisition and three-dimensional motor controller, etc.

3 Customizable Embedded Software Platform

The embedded software platform of intelligent sewing machine is mainly constructed by embedded OS, drivers for special devices and embedded application software, while some applications may need embedded graphic and character libraries, network

protocol stack and other modules. Traditional embedded software is difficult to be applied to the other intelligent sewing equipments, because it is often designed for special cases. To meet the seriate demands and improve the development method, a new customizable architecture of embedded software platform is proposed as following.

Such customizable software architecture is mainly separated into three different parts: customizable embedded OS, general embedded software component set, and special embedded software component set. With such architecture, the special application-oriented embedded system can be customized easily. During the development of new sewing machines, more embedded components can be developed and added into the embedded software set continuously. Eventually, the customizable embedded system platform with rich functions can be established, which will be more helpful in the future. At the same time, the "general software + special software" mode based on embedded component will meet the customizable demands.

3.1 Customizable Embedded RTOS

In our practice, we used embedded Linux as our foundational OS platform for it has more merits. Embedded Linux provides not only the core functions of the micro-kernel, such as general process management, but also the high-level ones, such as the real-time FIFO, process schedule based on priority-driven polling and otherwise. The architecture of embedded Linux is clear and can be divided into the following four layers: HAL, micro-kernel, real-time kernel and application interface. For the limited resources and domain-oriented applications, the common embedded Linux need to be tailored, extended and optimized before it is used in the sewing equipments. This work will be described as the following items.

Kernel Tailoring. Linux kernel is designed in modularization mode which allows many functional modules to be compiled into kernel directly or loaded dynamically. Because of the open source and powerful kernel-compiled tools, Linux kernel can be tailored conveniently. In addition, embedded Linux should also activate the kmod function to support diverse hardware by smaller core. When we try to tailor a kernel, one of the important contents is the compile-items. Based on our experiments, the following options must be considered: loadable module support, general kernel, block devices, networking options, ATA/IDE, Frame-buffer, input core, file system, SCSI and USB support, and etc. Additionally, we should consider tailoring the libc library according to application, which can economize the storage space by a long way.

For example, to realize the automatic network configuration and the pattern data transferring function, the items, such as Networking support [General setup], Network block device support [Block devices], and Unix domain socket, IP: kernel level auto-configuration, IP: DHCP support, IP: BOOTP support, IP: RARP support[Networking options], should all be selected.

In practice, we adopt the Linux 2.4.20-8 kernel. The kernel we customized for the electro-pattern sewing machine is about 1093K, which can support almost all the basic functions required by an electro-pattern sewing machine.

Function Extending. Generally, the drivers and interfaces provided by embedded Linux may not support all the devices needed by sewing equipment, especially some special hardware. Consequently, the embedded Linux kernel should be extended fully.

Linux kernel is designed as a wholly independent entity, and the drivers are all designed as modules. This mechanism allows all the drivers to be loaded or removed dynamically with some useful system calls, such as init_module( ) and cleanup_ module( ). Although the drivers are all treated as modules in Linux system, the kernel can load them by two different methods. One is modularization mode that allows the kernel loading special modules dynamically. And the other is kernel mode, which requires the kernel to load all modules needed when system is booting. Considering the fixed demands of the special sewing system, the latter is adopted in our practice.

Performance Optimizing. Embedded Linux can satisfy requirement of applications on sizes and functions partly, but its performance, such as real-time processing, may be limited if it is used in sewing field. Thus the performance of embedded Linux should be optimized firstly when it will be used.

- The ability of real-time processing

Because the interrupt mechanism and process scheduling policy in traditional embedded Linux are not designed in real-time mode, the real-time performance must be optimized for real-time sewing application. Based on the analysis, the real-time performance of Linux kernel can be promoted by modifying source code of kernel, adopting optimized process scheduling, inserting preemptable points, optimizing the mechanism of interrupt management and fining clock granularity [5].

- Rapid, graphic system booting

Another requirement of such embedded system is rapid system booting. For the sake of hardware detection and system services loading, common Linux always starts up slowly. But, not all of the detection and services are needed. To shorten the system booting time, some unnecessary system detections and services must be masked by modifying initial shell scripts and kernel configuration. On the other hand, Linux should be booted in the single user mode. Moreover, some applications also require the system booting in graphic mode, so that this function can be realized by patching the optimized kernel with bootsplash 3.0.7.

On an embedded board with 1GHz main clock and 128M memory, the optimized embedded Linux can boot up in 5 seconds, and shut down in 4 seconds, while the common Linux will spend about 73 seconds to boot and 24 seconds to halt in the same conditions.

3.2 Application Software Set Based on Embedded Component

Embedded components are self-contained, packaged, standard and reusable software modules with a special functionality [11]. Having analyzed the functions of intelligent sewing equipments, we divided embedded components into three classes: framework, general sewing components, and the special ones. This classification indicates the functional modes and the hierarchy of embedded component in sewing field, which is helpful to configure, extend, and manage the component library.

Embedded Component for Sewing Equipment (ECSE). An embedded component X can be described with data structure: component X::= <$I_R(X)$, $I_I(X)$, $I_E(X)$, $F(X)$, $I_H(X)$>, where $I_R(X)$ is the component remark, $I_I(X)$ is the data and control interface, $I_E(X)$ is the entity to realize the actual sewing function, $F(X) = \{f_1,\ldots,f_n\}$ is the set of services encapsulated by component X, $I_H(X)$ describes information about the related hardware. Considering the fact that the embedded component for intelligent sewing equipment is almost related to the control action, the descriptions of the time-restriction, hardware platform, special embedded interfaces and other features are supplied to the embedded component as metadata.

The procedure to design an embedded component mainly includes three steps: component abstraction, component entity design and component packaging. Twice-packing is necessary for component entity, and the first is to uniform the service interfaces while the second to manage the component library [9].

Embedded Component Repository for Sewing Equipment (ECRSE). ECRSE is a set of software modules constructed by certain semantic and structural rules. In ECRSE, all the embedded components are stored in the form of files, and all can be searched and customized by component management tools.

According to the classification of ECSE before, the ECRSE can be classified into three sublibraries.

- **Framework component subset** contains all the foundational components. These components are always the basic units, and provide abundant contracted interfaces.
- **General component subset** involves some components used by different equipments, such as power on self-test, data acquisition, auto-threadtrimming, thread-broken detecting and so on.
- **Special component subset** includes some components needed by special equipment, such as pattern customizing for electro-pattern-sewing machine, fastener hole diameter setting for electro-button-attaching machine.

ECRSE is established with information library and functionality library in practice. The information library stores the description information of all components, while the latter provides functionality to instantiate components. At the same time, the mapping relationship between description information and components is also necessary to be involved into ECRSE.

Customizing Application. Commonly, the process to customize an application involves system designing, components customizing and assembling, system simulating and testing, etc.

After analyzing and determining the characteristics and functionality of target system, the following step is to choose one appropriate framework, some general components and special ones for it. Since there aren't completely equivalent function descriptions in the component library where there aren't reduplicate components [10], new components should be customized or developed if no one can satisfy the requirement. The third, all the customized components should be assembled according to component contact and event/time-driven mode to form a new embedded application. By integrating the customized embedded software, RTOS and hardware platform together, the customization work is completed on the whole. As the last important step, the system customized above must be tested and simulated in the

special simulation environment we have developed. The principle of customizing component-based application is shown in Fig2.

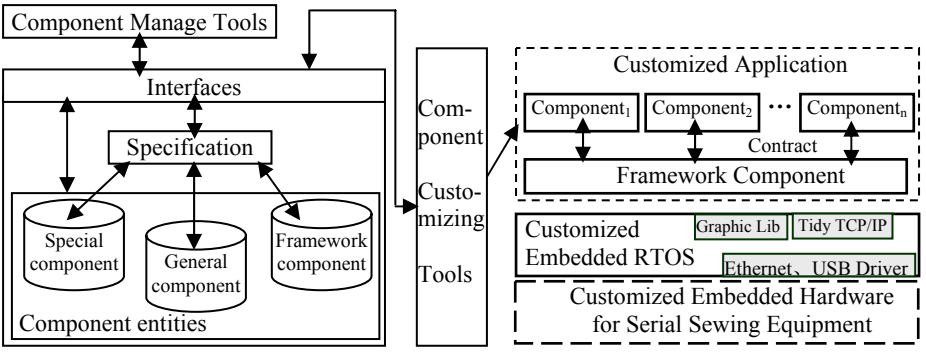


Fig. 2. Component-based Embedded Application for Seriate Intelligent Sewing Equipment

4 Conclusion and Future Work

As a whole, the opening system architecture for seriate sewing machines presented in this paper has been used in our project, and the result of experimentation shows that the customizable method is doable and effective. With our research and practice, we can get the following conclusion: the opening architecture and the component-based application development method are useful to shorten the developing period of a new product, and improve the performance of products further.

During our practice, we summarize two main tasks that should be researched more in the future. First, the existing kernel customization tools can't satisfy the special requirements, so we will study further the mechanism of customizable embedded RTOS and customization methods. In addition, we will optimize the interface and structures of embedded component according to the characteristics of seriate intelligent sewing equipment.

Acknowledgment

This work is supported by the National 863 Project under grant No.2004AA1Z2410 and Xi'an Science Technology Bureau under grant No.CH04004.

References

1. He Hai, Zhao Yanwen, Wu Shilin: The Design of Real Time Multi-task Controller for Home Computerised Embroidery Machine, Journal of WuHan Institute of Science and Technology, Vol.14, No.1(2001)18-21.

2. Diao Hongquan, Yan Gangfeng: Integral Design Scheme of Computerised Embroidery Machine's Control System, Journal of Engineering Design, Vol.10, No.4(2003)187-191.
3. Carsten Boke, Marcelo Götz, Tales Heimfarth: (Re-) Configurable Real-Time Operating Systems and Their Applications, The Fifth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems(2003)148.
4. Zhou Dexin, Zhang Xiangli: Embedded Operating System and Linux, Journal of Guilin Institute of Eectronic Technology, Vol.20,No.4(2000)21-23.
5. Yang Chaojun, Li Shichang, Tao Yang: The Optimization of Embedded Linux, Journal of Chongqing University of Postsand Telecommunications, Vol.14 No.4(2002)61-64.
6. Li Xiaoqun, Zhao Huibin, Ye Yimin, Sun Yufang: RFRTOS: A Real-Time Operation System Based on Linux", Journal of Software, Vol.14, No.7(2003)203-1212.
7. Yu Xianqing: Component, Component-Base, Method Based on Component, Journal of Graduate School, Academia Sinica, Vol.15, No.l(1998)86-90.
8. Philip T Cox, Song Baoming: A Formal Model for Component-Based Software, IEEE 2001 Symposia on Human Centric Computing Languages and Environments(2001)304.
9. Liu Yu, Guo Heqing: The Realization of Reusable Component for Special Domain, Micro-computer Applications, Vol.15, No.11(1999)21-23.
10. Zhang Haifei, Yuan Lei, Xia Kuanli: A Model of Software Component Libraries Function Set, Computer Engineering, Vol.26 No.11(2000)87-90.
11. Uwe E. Zimmermann, Michael Wenz, Thomas Längle, Heinz Wörn: Describing Components and Configurations for Embedded Control Systems. The Proceedings of the 4th International Workshop on Computer Science and Information Technologies CSIT'2002.

A Distributed Architecture Model for Heterogeneous Multiprocessor System-on-Chip Design*

Qiang Wu[1, 2], Jinian Bian[1], Hongxi Xue[1]

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China 100084
wuqiang2000@mails.tsinghua.edu.cn,
{bianjn, xuehx}@tsinghua.edu.cn
[2] College of Computer and Communication, Hunan University, Changsha, China 410082

Abstract. Current embedded system designs inspire the adoption of heterogeneous multiprocessor system-on-chip (SoC) technology, in which the architecture model plays a very important role. This paper proposes a distributed architecture model for the heterogeneous multiprocessor SoC design. It takes the view on the system as multiple processing elements connected with a network of communication channels. System functions are refined to primitives provided by the processing elements and communication channels through a hierarchy of abstraction layers. This will be helpful for the enhancement of system design modularity and efficiency.

1 Introduction

Current embedded systems design trend is to employ the heterogeneous multiprocessor System-on-Chip (SoC) technology to achieve functionality, performance, price and low power goals [1][2]. Chips summarized in [10] verify the emergence of such a design style, while [11], [12] and [13] give the concrete examples of this trend. In these designs, several microprocessors and ASIC blocks will be integrated on the same chip. Particular implementation tools are used for each of the components, e.g., compilers for microprocessors, synthesizers for ASICs. Communication components are often identified as hardware logics combined with software drivers, and implemented with corresponding compilers and synthesizers. Beyond them, the architecture model describes the organization of various components. It hides the details of the underlying components, exhibits a unified programming model for the designers to specify the system function, and provides an appropriate hierarchy of abstraction layers to pass the system function specification to back-end implementation tools.

* This work was supported by "National Natural Science Foundation of China 90207017, 60236020, 60121120706" and "Hi-Tech Research and Development Program (863) of China 2003AA115110".

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 150-157, 2005.

Intensive researches have been carried out on the embedded system design [3], but many of them focus on the single processor system, like COSYMA, VULCAN, LYCOS, TOSCA, MICKEY [3][4]. Some of them target at the multiprocessor system, like POLIS, COOL, Chinook, COSMOS, CoWare, SpecSyn [3][4] and MFSAM-based approach [5]. Among these, POLIS, COOL, Chinook, COSMOS were developed before 2000, and did not focus on the heterogeneous multiprocessor SoCs. CoWare and SpecSyn (and its successor, SpecC-based SCE [6]) consider the architecture of multiprocessor SoCs, but restrict themselves on some specific set of interconnect topology and communication protocols. MFSAM is closest to the focus of this paper which uses a generic architecture model for application-specific multiprocessor SoCs.

The work presented in this paper follows a distributed computing manner for the top view, regarding the system as multiple processing elements connected with communication channels. The main difference between this model and the previous works mentioned above is the manipulation of memories which are viewed as building blocks of processing elements or communication channels, not appearing at system level. This can maintain a clearer view at system level for an easy function-architecture mapping.

The rest of the paper is organized as follows: Section 2 describes the architecture model in detail. Section 3 outlines the translation of system functions through the hierarchy of abstraction layers. Conclusion is given in section 4.

2 Architecture Model

2.1 System Model

The proposed architecture model is a network of communication channels (CH) connecting multiple processing elements (PE), as illustrated in fig.1.

Memories are attached either to PEs as their local memories or to CHs as shared memories. No instances of independent memory exist in system level model of the architecture. This is different from previous works that always have memories as separate components in the architecture model, especially the shared memories. But in heterogeneous multiprocessor SoC designs, memories may have different structures for different processors. Such a separate memory organization in architecture model may have disadvantages for system scalability. MFSAM ameliorates this situation by integrating local memories with processors, thus enhances the modularity of the system. However, shared memories in MFSAM approach are still separated from other components in the system, especially from the communication network.

Considering that shared memories act as a center for information exchanges between different processes or tasks in the system, we think it is better to view shared memories as a communication channel. This forms a clean distributed computing model for the system, where PE performs the computation and CH performs the communication. We emphasize on the distributed feature of the model with the consideration of the relation to the back-end implementation tools. We hope that the compilers, synthesizers and programmers can be applied within the domain of each

PE and CH in the implementation stage. This encapsulation of individual components can increase the modularity and scalability of the whole system.

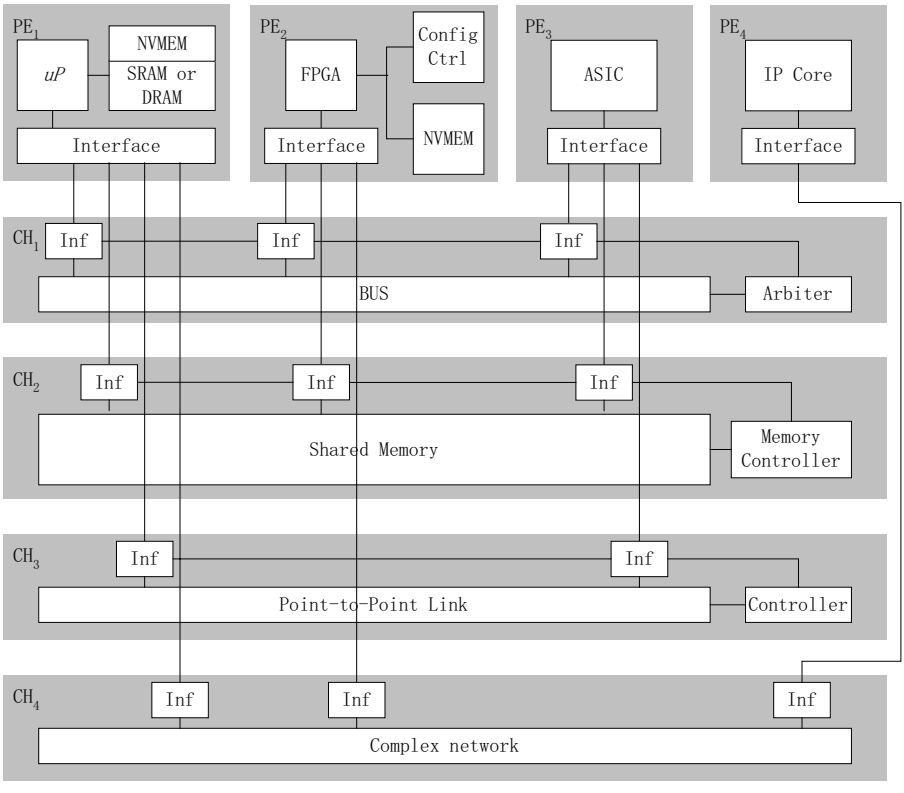


Fig. 1. Illustration of the distributed architecture model

It can be seen in the fig.1 that all PEs and CHs have interfaces between them. The interfaces on the PE side are used to hide the details of PEs and adapt them to the abstract communication model. The interfaces on the CH side are used to hide the details of CHs and provide an abstract communication model for PEs. This is helpful for a clear separation between computation and communication.

2.2 PE and CH Types

The PE can be of whatever computation component, such as microprocessor, DSP, ASIC block, reconfigurable logic device or IP core. They can be categorized to three types according to the programmability: (1) programmable PE (for microprocessors); (2) reconfigurable PE (for programmable logic devices); (3) fixed PE (for full-custom logic blocks), like PE_1, PE_2 and PE_3 illustrated in fig.1. IP cores that perform the computation of the system will be regarded as one of these three types of PE with the consideration of their programming characteristics and system needs. It should be

emphasized that each PE has a complete running environment for processes or tasks executed on it. Data exchanges and control synchronizations with other processes or tasks on different PEs are carried out through corresponding CHs.

The CH can be of whatever communication component, and are divided to four types with the consideration of topologic complexity: (1) point-to-point link; (2) bus; (3) shared memory; (4) complex network, as shown in fig.2 in detail.

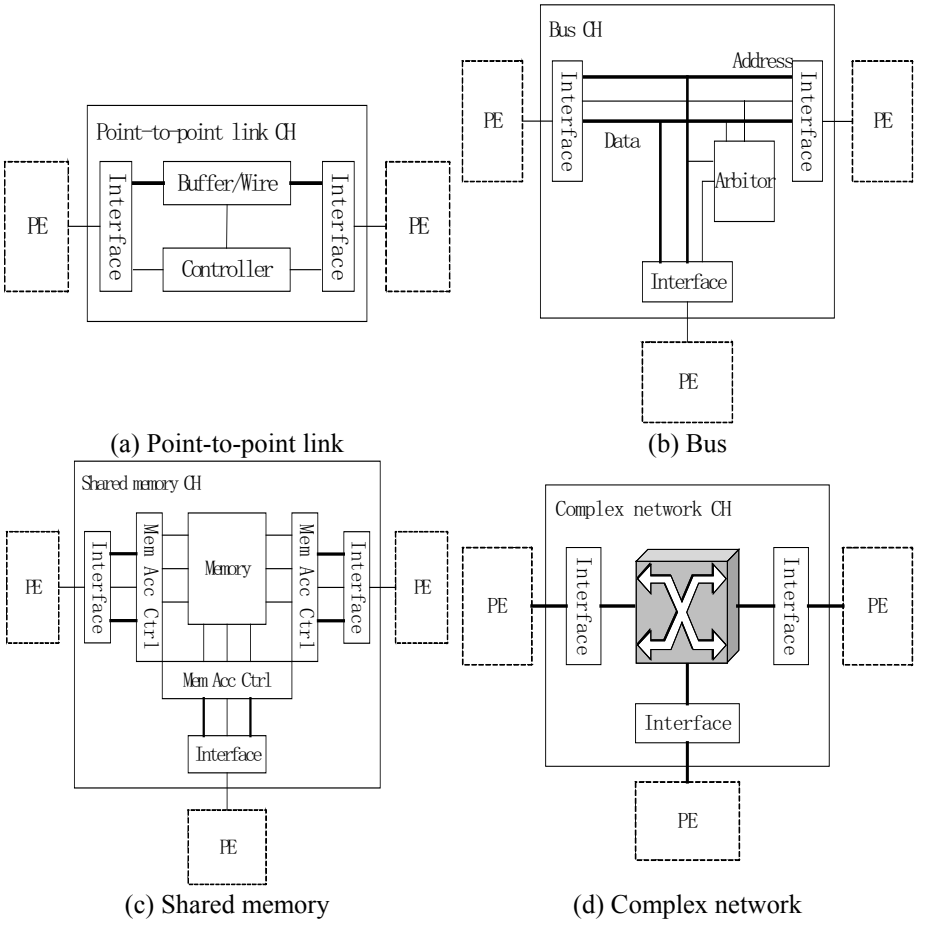


Fig. 2. Communication channel types

IP cores that perform the communication of the system are also classified in these four types according to their topologic characteristics.

Based on these fundamental types, PEs and CHs can be further divided to sub-types according to the functions they provide, as well as the performance and cost associated. Such type refinement can be carried on recursively, resulting in a type tree of PEs or CHs. The leaves of the tree should be the concrete PE or CH instances,

such as specific processors or buses. Theoretically, if all instances of PEs and CHs and all variations of their functions, performance and cost are considered, the type tree will be too large and possibly conflicting. This is not necessary and should be avoided in a practical system design. So, in real system designs, type refinement should be built on the complexity of the system function and the availability of hardware and software resources to select appropriate width and height of the type tree that can give efficient guidance to the system design.

For a good link to back-end tools, we suggest a principle of type refinement as follows: (1) At first, build sub-types from fundamental types by the functional compatibility; (2) Then, build further sub-types by the variations of performance and cost. This is to say, we group available PEs or CHs that can be alternated to implement the function of one or more processes in the system specification. After that, we will decide whether to divide them further or not with the consideration of system design requirement and the variances of performance and cost among the PEs and CHs in each sub-type. If the variance is not significant and can be ignored in system design decision, then the further division is not necessary. Otherwise, a deeper type tree is demanded.

Fig.3 gives an example of PE type trees. It consists of 2 sub-types of programmable PE: 8051 microcontroller and ARM microprocessor core series, 2 sub-types of fixed PE: JPEG codec core and DCT logic block and 1 sub-type of reconfigurable PE: FPGA fabric can be identified. The performance and cost differences of ARM9 and ARM9E should be examined for further consideration, indicated by two dotted lines. Usually, these two microprocessor cores need be separated to different sub-types. But cases may exist in some designs that the processes to be mapped on these two cores have minor differences in terms of design objects like running speed, power consumption, development cost and so on. In such a situation, the designer may figure that the choice between the two cores has little effect on the performance and cost of the final system, and decide not to distinguish them. This means the type tree ends at the ARM core series, not the core instances. For system design exploration and decision, this will be helpful due to the reduction of the search space, resulting in a faster design convergence procedure.

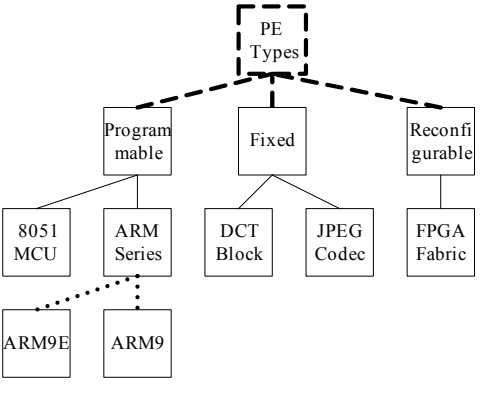


Fig. 3. Illustration of a PE type tree

The building of CH type tree is similar to PE type tree, while the compatibility consideration mainly concerns the interoperability of communication protocols and interface adaptability. And the terms of performance and cost often refer to the bandwidth, frequency, transport power, chip area and so on.

3 Design Flow

Unlike the top-down [7] and bottom-up [8] approaches, the design methodology in our work can be classified as the meet-in-the-middle approach. The whole design flow is shown in fig.4.

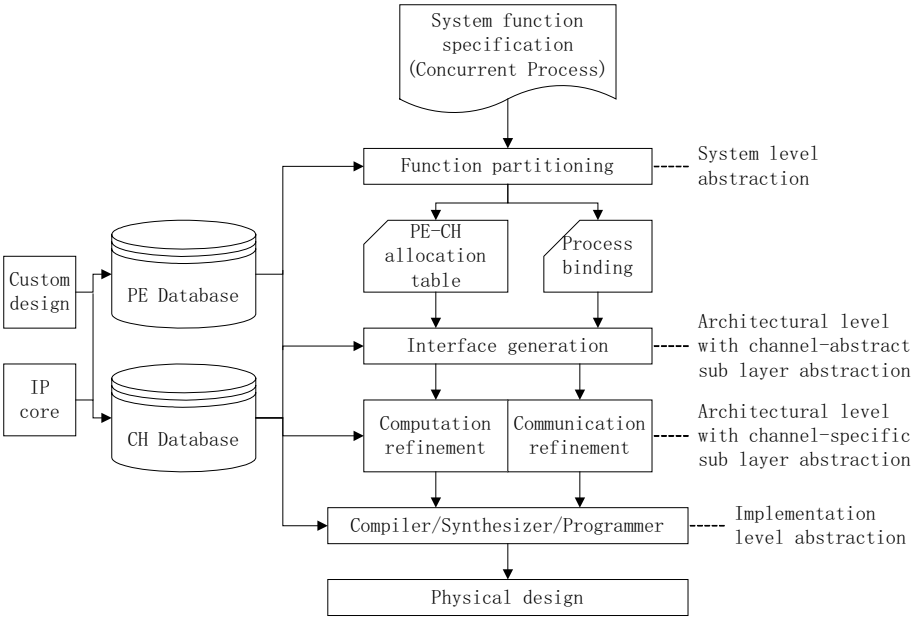


Fig. 4. Design flow

3.1 Abstraction Hierarchy

A three-layered hierarchy is adopted in our architecture model, which consists of the system level, architectural level and implementation level, as shown in fig.5.

At system level, the architecture model provides abstract computation and communication primitives to support system functional specification as concurrent processes. The computation primitives are abstract functions, while the communication primitives are based on the message passing mechanism.

At architectural level, computation is represented as the functions and statements in high level programming languages or hardware description languages (HDL). On the

PE side, communication is represented as operating system services with hardware dependent software drivers, or behavioral signaling in HDL. On the CH side, communication is further divided to two sub layers. The higher one is called the channel-abstract level, which provides communication primitives adapted to the PE's communication interfaces. The lower layer is called the channel-specific level, which describes the composition of the underlying hardware components for each channel.

At implementation level, computation and communication are merged to more basic operations in the forms of instruction set and logic functions. Compilers, synthesizers and programmers are introduced to take care of these tedious matters.

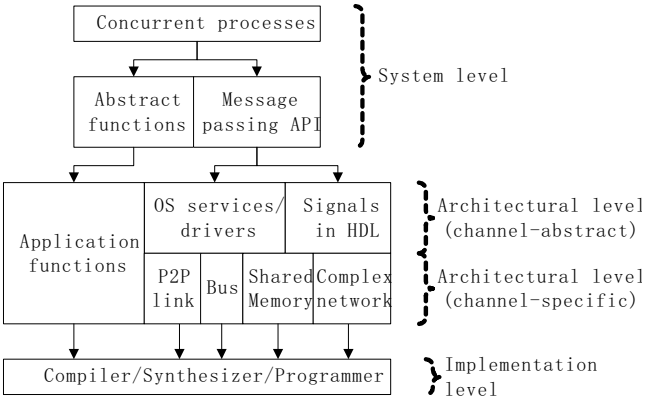


Fig. 5. Three-layered abstraction hierarchy

3.2 Meet-in-the-Middle Approach

At first system function will be analyzed and partitioned to groups of processes. Each group will be executed on the same PE. The communication channels will be attached to PEs having communicating processes. Then we build implementation instances for PEs and CHs in the preliminary architecture. Computation and communication abstraction will be performed for the PE and CH instances. The extracted primitives and parameters will be stored in PE and CH database for future reuse. Later we refine the partitioned processes towards the architectural level to meet these primitives. Once reached, the architectural model and parameters of the PEs and CHs will be used to guide the compilation and synthesis at implementation level.

Compared with a typical meet-in-the-middle method, the platform-based design [9], our design flow refines the system function to individual PEs and CHs rather than a platform. This is due to the distributed feature of our architecture model, which can provide designers more freedom in specializing PEs and CHs for a given application.

And our approach differs from component-based approach [8] in that the component-based approach focuses on the automatic generation of wrappers for components in the system to interact with each other, which mainly concerns the communication aspect of the implementation of system function. Our approach intends to cover both

computation and communication aspects of the system function to support architectural exploration, integration and implementation.

4 Conclusion

We have introduced a distributed architecture model and related design methodology. The model is featured with encapsulation of individual component and the separation of computation and communication. A hierarchy of system level, architectural level and implementation level abstraction is adopted in this model. Two sub abstraction layers in architecture level for the CHs are employed to assist the refinement of the communication interface between the PE and CH. This enhances the modularity and scalability of the system, which increases the design reusability and efficiency. A set of CAD tools is under development to support the design flow based on the proposed architecture model.

References

1. Flynn, M. Dubey, P. Hot chips 15 - scaling the silicon mountain. IEEE Micro, Vol. 24, Iss. 2 (2004) 7-9
2. Ravikumar, C.P. Multiprocessor architectures for embedded system-on-chip applications. Proc. of Intl. Conf. on VLSI Design, Bangalore (2004) 512-519
3. Staunstrup J, Wolf W. Hardware/Software Co-Design: Principles and Practice. Boston: Kluwer Academic Publishers (1997)
4. R. Rajsuman. System-on-Chip: Design and Test. London: Artech House Publishers (2000)
5. Amer Baghdadi, et al. An Efficient Architecture Model for Systematic Design of Application-Specific Multiprocessor SoC. Proc. of DATE, Munich (2001) 55-62
6. S. Abdi, et al. System-on-Chip Environment (SCE Version 2.2.0 Beta): Tutorial. Technical Report ICS-TR-00-47, University of California, Irvine (2003)
7. Lukai Cai, et al. Top-down system level design methodology using SpecC, VCC and SystemC. Proc. of DATE, Paris (2002) 1137
8. Cescirio W., et al. Component-based design approach for multicore SoCs. Proceedings of Design Automation Conference (2002) 789-794
9. Sangiovanni-Vincentelli, A., Martin, G. Platform-based design and software design methodology for embedded systems. IEEE Design & Test of Computers, Vol. 18, Iss. 6 (2001) 23-33
10. Michael Flynn, Pradeep Dubey. Hot Chips 15 – Scaling the Silicon Mountain. IEEE Micro, Vol. 24, Iss. 2 (2004) 7-9
11. Alireza Hodjat, Ingrid Verbauwhede. High-Throughput Programmable Cryptocoprocessor. IEEE Micro, Vol. 24, Iss. 3 (2004) 34-45
12. Deepu Talla, Ching-Yu Hung, Raj Talluri, et al. Anatomy of a portable digital mediaprocessor. IEEE Micro, Vol. 24, Iss. 2 (2004) 32-39
13. Hans-Joachim Stolberg, Mladen Berekovic, Lars Friebe, et al. HiBRID-SoC: a multi-core system-on-chip architecture for multimedia signal processing applications. Proceedings of the Conference on Design, Automation and Test in Europe: Designers' Forum - Volume 2 (2003) 8-13 suppl

A New Technique for Program Code Compression in Embedded Microprocessor[1]

Ming-che Lai, Kui Dai, Li Shen, and Zhi-ying Wang

School of Computer National University of Defense Technology, Changsha, P. R. China.
mingchelai@chiplight.com.cn

Abstract. Program code compression has become a critical technology nowadays, and it can promote the performance of microprocessor systems. But compression techniques nowadays can hardly achieve both satisfactory compression ratio and low hardware complexity. A new program code compression method is proposed in this paper. Experiment results show that the code sizes of ARM and OR1200 microprocessor can be efficiently reduced by 32.2% and 36.9% individually with this algorithm, resulting to a sharp decrease in memory traffic. A hardware decompressing prototype is also presented, revealing its advantage in ultra high speed decompression and low hardware complexity.

1 Introduction

With all kinds of the electronic equipments widely used on PDA and automobiles, embedded systems have been more and more popularized. Especially with the high development of information society nowadays, embedded systems are advancing towards large scale and complexity. Recent statistics show that the requirement for the embedded system memory is increasing linearly by year, but it is accompanied with a series of problems, such as shortage of storage resource, high price of the memory elements and so on. Thus, the research on the code compression technology is necessary.

The correlative research on the code compression begins at the nineties of the twentieth century, and there is an abundance of algorithms in the literature now. In 1992, Wolfe and Chanin [7] proposed the first code compression system for embedded processors using byte-based Huffman coding technology. They reported a compression ratio around 0.73 for MIPS code. The main shortcoming of this method is the low decompression speed. Lefury [4] proposed another new dictionary-based method and presented the concept of variable-length codeword. The compression ratios are 61%, 66% and 74% for the PowerPC, ARM and i386 programs on average respectively. The drawback of this method is that branch targets must be aligned and the range of branched is reduced. IBM also presented a code compression method named CodePack [1], which split 32-bit instruction into two 16-bit words and compressed the program using the variable-length codeword. They reported that the

[1]Supported by the Natural Science Foundation of China under Grant No.60173040

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 158-164, 2005.

compression ratio could be achieved about 61% for PowerPC program code. Additionally, there are some new code compression methods based on the new architecture, such as MIPS16 [6] and ARM Thumb [5]. The ARM Thumb instruction set consists of a subset of the most commonly used 32-bit ARM instructions. The ARM Thumb instruction set achieves an average compression ratio of 70% while MIPS16 achieves about 60%. They have a zero decoding penalty but result in the loss of the performance and require more effort on the microprocessor design.

This paper presents another new program code compression algorithm in section 2. The related hardware implementation is discussed in section 3. Section 4 presented the corresponding experiment results.

2 Code Compression Algorithm

2.1 Basic Algorithm

In order to achieve a satisfactory compression ratio by real-time decompression, this section present a new program code compression technique, which is referred as PCCP for Program Code Compression based on the Pretreatment.

First of all, a pretreatment mechanism is introduced. The aim of this mechanism is to make the uncompressed code bits with value 0 become clustered relatively. Before the pretreatment, here is a matrix whose rows correspond to the uncompressed code vectors. Using this matrix, the pretreatment scheme works in the following two steps:

1. By studying all the elements of each column in the matrix, the corresponding bit of the vector X_i is made certain as follows: If the number of the element 0 exceeds the number of the element 1 in a certain column, the corresponding bit of the vector X_i will be set to 1. Otherwise, it will be set to 0.
2. Doing XOR operation between X_i and the former code vectors to get the new code vectors for the rows of the new matrix. After replacing the old matrix, go back to the step 1 to get the new vector X_{i+1}.

The two steps above will be repeated P times, and a new code vector β will be generated for each input vector α. That is to say, $\alpha=\beta \oplus X$, where X represents $X_1 \oplus X_2 \cdots \oplus X_p$. It is assured that α can be recovered because β and X are both known.

After the pretreatment, the new code compression algorithm is presented. The compression scheme works in the following four steps:

1. Get the new matrix and the vector X by the above pretreatment. And each row corresponds to an uncompressed code vector.
2. Select a maximal vector from the matrix as the next dictionary item T_i.
3. For each code vector Y labeled as uncompressed,

$$Y=T_i-Y \tag{1}$$

Then a new matrix can be generated, in which any code vectors smaller than the threshold θ is labeled as compressed.

4. If all the code vectors in the new matrix are compressed, the compression finishes. Otherwise, go to step 2.

Completing the above procedure, the dictionary has been generated. But it is not suitable for the decompression. In fact, this dictionary only needs some mathematical

improvement to achieve the simple implementation and fast decompression. Suppose there is an uncompressed program code sequence $x_1, x_2, \cdots, x_n$. Following the above scheme, this sequence becomes another one $y_1, y_2, \cdots, y_n$ ($\forall i$ (y_i<θ)) while getting the dictionary items $T_1, T_2, \cdots, T_n$. If a certain code vector x_i is compressed to the vector y_i by some dictionary items $T_1, T_2, \cdots, T_w$. There exists the following conclusion:

$$T_w - (T_{w-1} \cdots - (T_1 - x_i)) = y_i \tag{2}$$

Then,

$$T_w - T_{w-1} - \cdots (-1)^{w-1} T_1 - y_i = (-1)^{w-1} x_i \tag{3}$$

$$t_w - y_i / (-1)^{w-1} = x_i \tag{4}$$

$$t_w = (T_w - T_{w-1} - \cdots (-1)^{w-1} T_1) / (-1)^{w-1} \tag{5}$$

In the formula (3), the code vector x_i uses the first w dictionary items for its compression. So the fast decompression can be realized only using the formula (4) and (5). If t_w is treated as the new dictionary items, there exists the following formula:

$$x_i = t_w + (-1)^w y_i \tag{6}$$

Then, using the new dictionary items t_w, the compressed vector y_i can be decompressed to the original code vector x_i.

2.2 Dictionary Size

A key factor of the compression is the dictionary size. Any good code compression method should limit the dictionary size. The scheme in the PCCP to reduce the dictionary size works as follows: if the value of δ' is less than the threshold δ^*, which is determined by the dictionary size and the constraint size, the item t_w is deleted.

$$\delta' = \sum_{j=1}^{p} \delta(x_{i_j}) - \sum_{j=1}^{p} \delta(y_{i_j}) - \delta(t_w) \tag{7}$$

Then, the models of the PCCP algorithm can be constructed immediately. In the models, a self-adapt recursive search strategy is adopted to satisfy the dictionary size constraint and to gain the best compression performance by selecting the proper threshold θ and δ^*.

3 Hardware Prototype

In the implementation of the PCCP, the biggest challenge is the effective run-time decompression and the key is the address conversion. Ideally, solving these problems only needs a bitmap to show whether each instruction is compressed or not. However, the main shortage is that the hardware will be a burden when the program is large enough. Thus, a block mechanism is adopted in the design to simplify the hardware

for the address conversion. As shown in Fig. 1, 128 sequential instructions will be organized as a block, whose correlative information item contains a bitmap and its base address. Using an item of the Address Conversion Table (ACT), any compressed instruction in the corresponding block can be accessed normally by the processor.

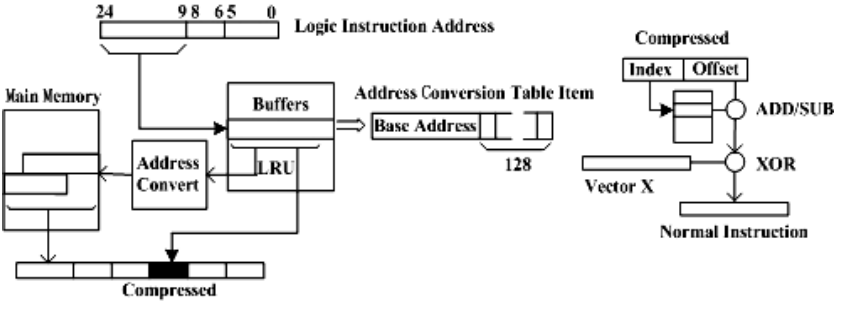


Fig. 1. The fast decompression in PCPP

As shown in the left part of the Fig. 1, the instruction address is decomposed into three parts. The sixteen most significant bits are used for the ACT index, and they are compared to the tags held in the buffers. Once the tag matches an entry and there is a cache miss, the corresponding ACT entry is used for address conversion. If the ACT entry is not in the buffers, it will be read from memory using the ACT index. The refill continues after the ACT entry is present. Then, if the compressed cache line is fetched from the main memory, the decompression unit will decide any an instruction to be decompressed or not, according to the corresponding bit in the bitmap. In the right part, the compressed instruction is split into two parts, the high significant bits are used as the dictionary index, and the low significant ones are used for the offset. The PCPP only needs to do SUB or ADD operation between them to recover the instruction (an XOR operation may be added because of the pretreatment).

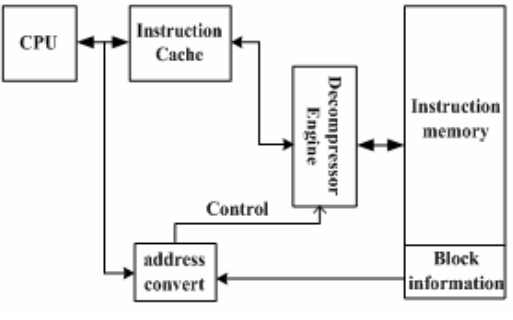


Fig. 2. Overall memory system organization

Then, Fig. 2 shows the simple instruction memory hierarchy in a typical system. From the analysis above, the logic of the address conversion is very easy, and its controller only consumes around **1.1k** gates in the following experiment on the ARM architecture. Then, the process of its address conversion just needs **2 ns** in the 0.18us

process. Therefore, it can be concluded that the decompression of the PCCP is real-time and only needs little hardware cost for the implementation.

4 Experiments

Finally, the experiments are performed on ARM and OR1200 architectures and the corresponding compression results are analyzed in the below.

Table 1. The compression result on the ARM architecture

No.	Program Name	Original Code Size	Dictionary size	Compression Ratios
1	ammp	270.2KB	1678	0.678225
2	go	422.7KB	1888	0.635728
3	mcf	60.0 KB	994	0.703374
4	gzip	87.6 KB	890	0.709907
5	vortex	618.0KB	1854	0.673422
6	vpr	207.3KB	1750	0.709019
7	bzip2	82.9 KB	830	0.706707
8	twolf	269.0KB	1872	0.708113

All the simulations performed in the Table 1 use a number of commonly available programs in SPEC 2000. For better compression performance in PCCP models, the length of the compressed instruction is chosen to be 16 bits. As a result, the ratios of compressed program size (including dictionary) over original program size in each benchmark distribute between 59% and 67% with a constraint that the dictionary size can't exceed 8 KB. But the overhead of the ACT is approximately 3.61% of original program size. So the inductive conclusion from the result shown in the Table 1 is that the compression ratio is approximately **67.8%**. With the same constraint, the compression ratio on the OR1200 architecture achieves **63.1%**.

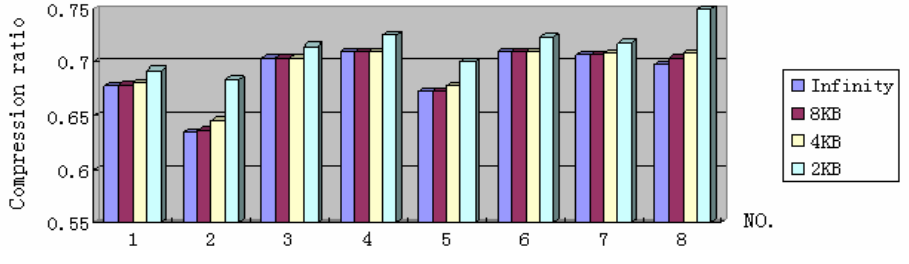


Fig. 3. The compression ratio under different dictionary constraints

Then, the size constraint of the dictionary is set to infinity, 8KB, 4KB and 2KB individually on the experiment. Fig.3 shows four bars representing the different compression ratio in the different size. The first bar is so similar to the second and the third one that the dictionary constraint size beyond 4KB seems to have little influence to the compression ratio. But 2KB size constraint is quite different. The compression

ratio has a relative increase here. So **4KB** is chosen for the dictionary size in the implementation at last.

With the dictionary size of 4KB, the experiments are performed on the ARM architecture again. The configuration of its instruction cache is 16-Kbyte, 1-way direct mapped instruction cache with 64-Bytes per line. The statistics in the Table 2 is about the memory traffic caused by the program code. And the result shows that there is a sharp decrease in the traffic of its 32-bits bus because of the compressed code.

Table 2. the memory traffic caused by the program code

No.	Program Name	Memory traffic without code compression	Memory traffic with code compression
1	ammp	6.21 E+09 Bytes	4.39 E+09 Bytes
2	go	1.39 E+09 Bytes	1.10 E+09 Bytes
3	mcf	2.07 E+09 Bytes	1.35 E+09 Bytes
4	gzip	2.13 E+10 Bytes	1.37 E+10 Bytes
5	vortex	2.60 E+10 Bytes	1.86 E+10 Bytes
6	vpr	5.14 E+09 Bytes	3.31 E+09 Bytes
7	bzip2	9.60 E+09 Bytes	6.23 E+09 Bytes
8	twolf	1.81 E+10 Bytes	1.17 E+10 Bytes

5 Conclusions and Future Work

This paper presents a new code compression algorithm for embedded system. As mentioned above, the standard to evaluate a code compression method can't be limited to the compression ratio. The other key factors should be paid more attention to, i.e. the compression ratio, the limited hardware cost, the real-time decompression, the small additional table, the low dependence to the processor and so on. Compared with Lekatsas [8] and other excellent code compression methods as mentioned above, the PCPP has an attractive compression ratio and little influence to the processor itself. Besides these, it decompresses in the real time only with low hardware cost.

The following related problems are to be studied further. The first is to apply the PCCP to the VLIW architecture [3] and evaluate the performance. The second is to study on the power impact to attack the problem of power consumption in the future for better performance [2].

References

1. T. Kemp, R. Montoye, J. Harper, J. Palmer, and D. Auerbach. A Decompression Core for PowerPC. IBM Journal of Research and Development, Vol. 42(6):807–812, November 1998.
2. Ismail Kadayif and Machmut T.Kandemir, "INSTRUCTION COMPRESSION AND ENCODING FOR LOW-POWER SYSTEMS" 15th Annual IEEE International ASIC/SOC Conference, pp.301-305, 2002.

3. Y. Xie, W. Wolf, and H. Lekatsas. A Code Decompression Architecture for VLIW processors. Proceedings of the 34th Annual International Symposium on Microarchitecture, pages 66–75, December 2001.
4. C. Lefurgy, P. Bird, I.-C. Chen, and T. Mudge, “Improving code density using compression techniques”, Proceedings of the 30th Annual International Symposium on Microarchitecture, December 1997.
5. K.D. Kissell. MIPS16: High Density MIPS for the Embedded Market. Silicon Graphics Group,1997.
6. Advanced Risc Machines Ltd. An Introduction to Thumb. March 1995.
7. Wolfe and A. Chanin, “Executing Compressed Programs on an Embedded RISC Architecture,” International Conference on Microarchitecture, 1992.
8. H. Lekatsas and W. Wolf, “Code compression for embedded systems,” in ACM/IEEE Design Automation Conf., San Francisco, CA, June 1998, pp. 23–28.

Design of System Area Network Interface Card Based on Intel IOP310

Xiaojun Yang[1, 2], Lili Guo[1], Peiheng Zhang[2], and Ninghui Sun[2]

[1] School of Information and Communication Engineering, Harbin Engineering University, Harbin 150001, China
guolili@hrbeu.edu.cn
[2] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China
{yxj, zph, snh}@ncic.ac.cn

Abstract. A design of system area network interface card (NIC) based on the Intel IOP310 I/O processor chipset is proposed in this paper. The chipset makes it powerful for the NIC to offload the processing of communication protocol from the host CPU. A network interface unit (NIU) based on memory bus is embedded in the NIC. The NIU not only thoroughly compensates for the lack of high performance data transfer channel in the embedded system, but also efficiently utilizes the memory bus bandwidth and direct memory access (DMA) engine to reduce the latency for data transfer between the host and network. The NIC is a part of DCNet, which is the system area network (SAN) of Dawning 4000A Cluster[1]. The testing results of DCNet show that the NIC obtains competitive communication performance compared with Myrinet, SCI, and QsNet, and prove that the way to design high performance NIC is feasible.

1 Introduction

With the enlargement of supercomputer scale and the improvement of node performance, the demands for bandwidth, latency and reliability of system area network (SAN) are becoming more and more important. At present, the SAN used in cluster architecture supercomputer is usually SCI [1], Myrinet [2], and QsNet [3].

An effective network interface card (NIC) is the critical aspect for a SAN to achieve high performance. The NICs of the above three SANs are all based on a SoC (System on a Chip) approach. However, with the development of embedded technology, more and more embedded systems have particular advantages such as processing capability, I/O, cost, and potential to be improved. Now it is possible to design a high performance NIC using universal embedded system. Based on the Intel IOP310 I/O processor chipset, a NIC is designed for DCNet, which is the SAN of Dawning 4000A Cluster. In order to achieve the high bandwidth and low-latency of the com-

[1] This research was supported by the National High-Tech Research and Development Plan of China under Grant, No. 2002AA104410.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 165-171, 2005.

munication, a network interface unit (NIU) based on memory bus is embedded in the NIC.

The remainder of the paper is organized as follows. Section 2 briefly introduces the NIC and the Intel IOP310 I/O processor chipset. Section 3 details the design of DCNet NIC. Section 4 presents the results of the performance evaluation. Finally, section 5 concludes the paper and discusses the future work.

2 Related Work

In this section, we make overviews of the NIC technology and the Intel IOP310 I/O processor chipset.

2.1 NIC Technology Overview

SAN is the high performance interconnection for a supercomputer. It consists of the switch, NIC, and communication protocol. Summary of the above three interconnects is given in Table 1 [4]. The table shows that SAN possesses high performance in bandwidth and latency. It is essential to construct the high performance SAN.

Table 1. Summary of SAN specifications and estimated cost

SAN	Bandwidth (MB/s)	Latency (μs)	Cost/Port (Euro)
QsNet (Quadrics)	360	5	4770
Myrinet (Myricom)	245	7	2050
SCI (Dolphin) 2D 5×5	200	4	1590

The processing and transferring capability of hardware is an increasing requirement of a NIC. It needs a high-speed processor and a low-latency data access channel to process the communication protocol. It is necessary to have a high bandwidth data transfer channel and an effective transfer mode (e.g., DMA) in order to assure the data transfer performance. Today's SANs such as SCI, Myrinet and QsNet all have a dedicated I/O processor and onboard memory, which offloads the protocol handling from the host CPU and ensures that all available PCI bandwidth is dedicated to data communication. They all support DMA and use PCI bus [5].

2.2 Intel IOP310 I/O Processor Chipset Overview

The Intel IOP310 I/O processor chipset with Intel XScale technology is the first product in Intel's fourth-generation of I/O processors. It contains two devices: the Intel80200 processor based on Intel XScale microarchitecture [6] and the Intel80312 I/O companion chip [7]. The chipset brings a dramatic increase of I/O performance. When used as an add-in card, the chipset provides the benefit of offloading the processing of the communication protocol from the host CPU.

The Intel80200 processor is compliant with the ARM Version 5TE instruction set. It has the good performance for moving and processing large amount of data quickly, and hiding memory latency. The Intel80312 I/O companion chip is designed to work with the Intel80200 processor to provide a cost effective solution for the intelligent add-in cards. It is a multi-function device that integrates the PCI-to-PCI Bridge Unit, DMA Controller, Integrated Memory Controller, and Application Accelerator Unit (AAU), into a single system chip.

3 Design of NIC Based on Intel IOP310 I/O Processor

In this section, we illuminate the design of a NIC based on the Intel IOP310 I/O processor chipset. The architecture of NIC, the data transfer modes, and the control program of NIC are illustrated.

3.1 Architecture of NIC

The Intel IOP310 I/O processor chipset lacks a high performance data transfer channel between the host and network, which is absolutely necessary to a high performance NIC. In order to solve the issue of the universal embedded system, as shown in the Figure 1, a network interface unit (NIU) based on memory bus is embedded in the NIC. The NIU efficiently utilizes the memory bus bandwidth and DMA engine to implement a high performance data transfer channel between the host and network. Figure 1 shows the function block diagram of NIC.

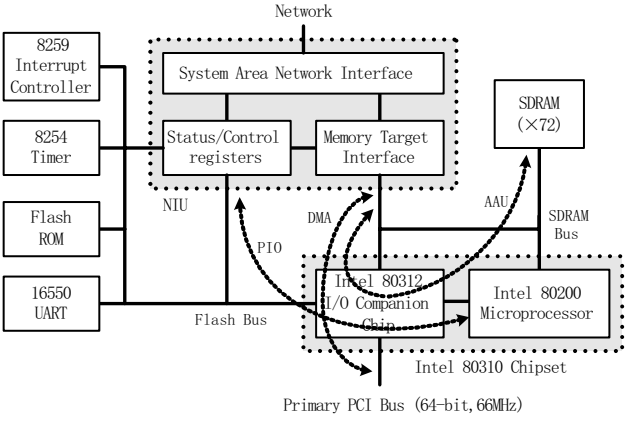


Fig. 1. DCNet NIC architecture

The NIC includes Intel80312 Companion Chip, Intel80200 Processor, 64MB Memory, 8MB FLASH, 8259 Interrupt Controller, 8254 Interval Timer, 16550 UART, and the NIU. This architecture not only offloads the communication protocol processing, but also supports two DMA engines between the host and network.

The NIU consists of Memory Target Interface (MTI), Status/Control Registers (SCR), and Network Interface (NI). The MTI is a virtual memory device. It converts the memory bus [8] to a simple local bus, which is used to communicate with the NI. The SCR is the NIU operation registers based on FLASH bus. The Intel80200 processor can access these registers. The NI performs the physical and data link level protocols. It is the same for the memory controller to access the MTI and the local memory. The data that the memory controller has written to the MTI will be sent to network, and the data coming from network is stored in the MTI and then will be taken away by the read operation of the memory controller.

Table 2 describes the NIC memory map. The address range of SDRAM $Bank_1$ dedicates to the MTI of NIU. The address range of Flash $Bank_0$ dedicates to the SCR of NIU and peripherals such as the UART and timer.

Table 2. The NIC memory map

Address Range	Layout
FE9EFFFF to FE800000h	SCR of NIU and On-Board Devices (Flash $Bank_0$)
FE7FFFFF to A4000000h	MTI of NIU (SDRAM $Bank_1$)
A3FFFFFF to A0000000h	SDRAM $Bank_0$
9FFFFFFF to 90020000h	Reserved
9001FFF to 80000000h F	ATU Outbound Transaction Windows
7FFFFFFF to 00800000h	ATU Outbound Direct Addressing Windows
007FFFFF to 00002000h	Flash $Bank_1$
00001FFF to 00001900h	Reserved
000018FF to 00001000h	Peripheral Memory Mapped Register
00000FFF to 00000000h	Initialization Boot Code From Flash $Bank_1$

3.2 Data Transfer

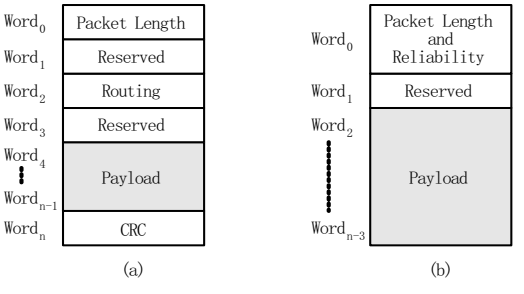


Fig. 2. Packet definition. (a) Sending packet and (b) Receiving packet

The packet length used for the transaction in a SAN is unlimited. The cell of a packet is a word (32bits). The packet definition is shown in figure 2, the sending packet header carries the packet length and routing information, which will be thrown away by the NIC and switch in the transaction processing, respectively. The receiving

packet header consists of the state information, which includes length and CRC check result of the receiving packet. The CRC field is used for assuring the reliability of the packet when received. The payload filed is the valid transfer data between the nodes.

The Intel IOP310 I/O processor chipset provides three data transfer modes: DMA, AAU, and PIO. The Intel80312 provides two DMA channels that perform the data transfer between the primary PCI bus and the local memory of the Intel IOP310 I/O processor chipset. AAU performs the low-latency, high-troughput data transfer to and from the local memory. PIO is the low-speed access mode that the Intel80200 accesses the devices located on FLASH bus. Because the NIU is a virtual memory device, the above three data transfer modes can be used all. A view of data transfer between the host and the NIU in the embedded system is presented in Figure 1.

3.3 Control Program of NIC

The NIC achieves all functions of a SAN, such as parallel communication, reliable and ordered communication, and timeout and resend mechanism. The control program of NIC (NCP) is a significant part of overall communication protocol. It runs on the NIC, and controls all the NIC operations for message passing. The design of NIU makes the structure of NCP very simple. As shown in Figure 3, the NCP operation for message passing is based on an event mechanism.

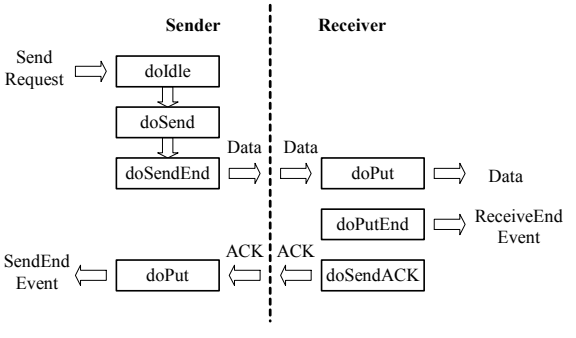


Fig. 3. NCP operation for message send/receive

4 Performance Evaluation

The testing platform consists of 8 servers (dual Intel Pentium III 1.0GHz CPU, 1GB memory, 64-bit/66MHz PCI bus, and Linux 2.4.18-SMP OS), 8 DCNet NICs and an 8-port DCNet switch. We test the hardware performance and the user-level communication performance.

The hardware performance of the NIC includes latency and bandwidth: 1.9μs in ping-pong latency for small message and 333MB/s in maximal ping-ping bandwidth. The testing result shows that the performance of hardware implementation based on universal embedded system and the NIU approach is enough for a SAN.

The user-level communication performance is the actual application performance of a NIC. The testing results show that DCNet achieves 14.5μs in user-level ping-pong latency and 218.0MB/s in user-level ping-ping bandwidth. The user-level communication performance of DCNet is close to that of Myrinet, SCI, and QsNet, which are based on a SoC approach. Furthermore. MPI programs correctly run on the 8 nodes platform based on DCNet. The BER performance of the NIC is less than 10^{-15}.

The testing results show that the NIU is a reasonable solution to design high performance NIC based on the low-cost and universal embedded system.

5 Conclusion and Future Work

In this paper, we have proposed the design of a NIC based on a universal embedded system, which is the key technology of Dawning 4000A DCNet. The implementation of the NIC obtains the reasonable user-level communication performance of DCNet. Though the testing results of DCNet show that the NIC obtains unperfect user-level communication performance compared with Myrinet, SCI, and QsNet, they prove that the design of high performance NIC based on universal embedded system and the NIU approach is feasible and effective. As shown in Figure 4, the performance problems of NIC will be readily solved with the application of other high performance embedded systems such as Intel I/O processors [9].

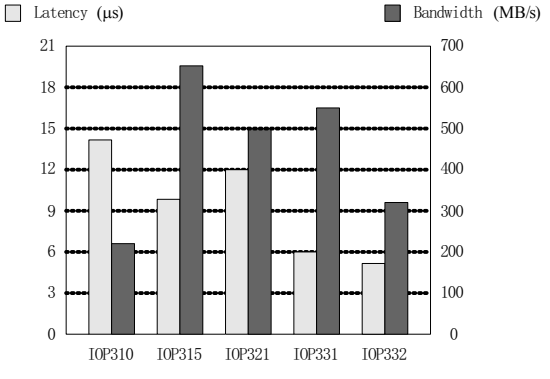


Fig. 4. Performance prospect of NIC based on Intel I/O processors

The future work includes adding some functions to the NIU to reduce the latency, and designing the next generation NIC based on the latest Intel I/O processors. The performance of NIC based on universal embedded system will be close to that of NIC based on the SoC approach more and more.

References

1. High Speed Network and Interconnect Products. http://www.dolphinics.com/. August 2003.
2. Myrinet Products. http://www.myri.com/. August 2003.
3. QsNet High Performance Interconnect. http://www.quadrics.com/. August 2003.
4. Cluster Design. http://www.clustervision.com/cluster_design.html. August 2003.
5. PCI Local Bus Specification, Revision 2.2. December 18, 1998.
6. Intel Corporation: Intel80200 Processor. November 2000.
7. Intel Corporation: Intel80312 I/O Companion Chip. December 2000.
8. Intel Corporation: PC SDRAM Specification, Revision 1.7. November 2000.
9. Intel I/O Processors. http://www.intel.com/design/iio/. October 2004.

Dual-Stack Return Address Predictor[1]

Caixia Sun[1] and Minxuan Zhang[2]

[1] College of Computer, National University of Defense Technology,
Changsha 410073, Hunan, P.R.China
cxsun1979@163.com
[2] College of Computer, National University of Defense Technology,
Changsha 410073, Hunan, P.R.China
mxzhang@nudt.edu.cn

Abstract. Return address predictors used currently almost have the same architecture: a return address stack and a top-of-stack pointer, some of which may be enhanced by repair mechanisms. The disadvantage of this type of return address predictor is that either prediction accuracy is low or the hardware cost is high. In this paper, we present a novel kind of return address prediction structure called Dual-Stack Return Address Predictor (DSRAP) which contains two return address stacks: RAS_PRED and RAS_WRB. Just as the return address stack in current return address predictors does, RAS_PRED provides predicted target addresses for procedure returns. RAS_WRB provides data for repairing RAS_PRED when a branch misprediction is detected. Results show that DSRAP can acquire 100% hit rates if mispredictions caused by unmatched call/return sequences or the stack overflow are ignored. Furthermore, DSRAP is very easy to design.

1 Introduction

Branch mispredictions have become a serious bottleneck to better performance for microprocessors with wide-issue and deep-pipeline. Each misprediction results in several, even more cycles of pipeline stalls. The target address misprediction of procedure returns is one important source of mispredictions, because a procedure may be called from many different locations, while the target of a particular return varies. If we use the old target address as the predicted target of current return instruction, misprediction might occur. Therefore, a special prediction structure is needed to deal with procedure returns. A return address stack [1, 2] is a good choice to provide target address for each return instruction. Most current microprocessors include such a return address stack. For example, in Alpha 21164, there exists a twelve-entry return address stack [3], and in Alpha 21264, a thirty-two-entry one [4]. Many of Intel's processors, including Itanium [5] and Itanium2 [6], also include a return address stack.

[1] This work was supported by Chinese NSF project (60376018) and Chinese NSF project (90207011).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 172-179, 2005.

A simple stack, however, fails in the presence of speculative execution. In the branch prediction stage, branch instructions are predicted and subsequent instructions, which may include calls and returns, are fetched speculatively. If a misprediction is detected later, all instructions executed speculatively should be squashed. Calls and returns update the return address stack in the branch prediction stage, while mispredictions are detected in the writeback stage, and as a result, calls and returns on a wrong path corrupt the return address stack. Not to undo the effects of squashed instructions, return address misprediction might occur later. So a simple return address stack cannot satisfy high-performance microprocessors that allow speculating.

In this paper, we propose a novel kind of return address prediction structure. We call it Dual-Stack Return address Predictor (DSRAP) for that it comprises two return address stacks. DSRAP can undo the effects of squashed instructions on the return address stack and achieve 100% return address prediction accuracy if mispredictions caused by the stack overflow or unmatched call/return sequences are ignored.

The remainder of this paper begins with a brief discussion of related work. In section 3, the architecture of DSRAP is described, and some design issues are discussed. The experimental environment and benchmarks are presented in section 4. In section 5, the simulation results are explained. Conclusions are given in section 6.

2 Related Work

Return address predictors used currently have the same architecture: a return address stack and a top-of-stack (TOS) pointer. We call this kind of prediction structure Single-Stack Return Address Predictor (SSRAP). In order to achieve higher return address prediction accuracy, SSRAP may be enhanced by different repair mechanisms. There are mainly three kinds of repair mechanisms used widely.

The easiest repair mechanism is to save the current top-of-stack pointer each time a branch is predicted [7]. We call this scheme saving the TOS pointer in the rest of this paper. A copy of the current TOS (C_TOS) is associated with each branch instruction that is detected in the branch prediction stage. When a branch misprediction is detected, the associated C_TOS is adjusted according to the type of branch misprediction and the TOS pointer is updated with the adjusted C_TOS. Saving the TOS pointer is very cheap. However, when there exist instruction sequences like a return followed by a call on the mis-speculated path, the return address stack would be corrupted.

A more aggressive scheme is to also save the content of the top stack entry along with the TOS pointer [8]. We call this scheme saving the TOS pointer and TOS content in the rest of this paper. This method can improve prediction accuracy greatly, but it will still fail if two returns are followed by a call on the mis-speculated path.

The extreme is to save the entire return address stack at the time of each branch prediction. We call this scheme saving the entire stack in the rest of this paper. For this scheme can always undo the effects of squashed instructions on return address stack, it can achieve 100% prediction accuracy unless unmatched call/return sequences exist or the return address stack overflows. Saving the entire stack, however, is very expensive, so it is employed rarely.

3 Dual-Stack Return Address Predictor

3.1 Architecture

Different from SSRAP, Dual-Stack Return Address Predictor proposed here includes two return address stacks, as shown in Fig.1. Just as the return address stack in current return address predictors does, RAS_PRED provides predicted target addresses for procedure returns. RAS_WRB provides data for repairing RAS_PRED when a branch misprediction is detected.

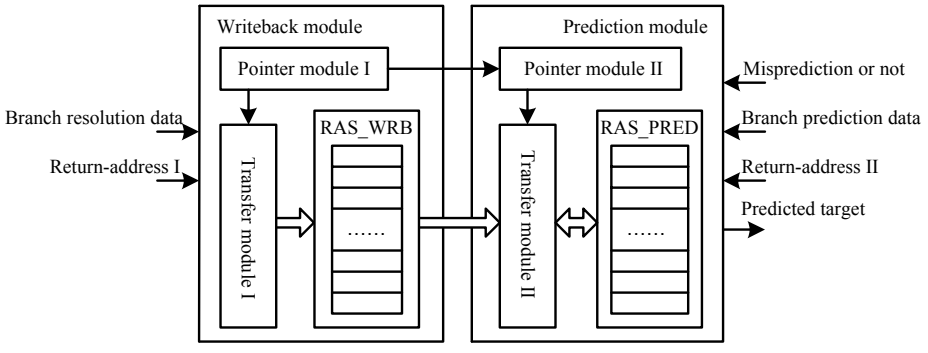


Fig. 1. The Architecture of DSRAP. Branch prediction data include the type of a particular branch instruction and its predicted direction. Branch resolution data include the type of a particular branch instruction and its actual direction. Return-address II is the return address of the call in the branch prediction stage. Return-address I is the return address of the call in the writeback stage

RAS_PRED and RAS_WRB have the same structure and the same number of entries. Two top-of-stack pointers, TOS_PRED and TOS_WRB, are defined to index the two stacks. TOS_PRED is for RAS_PRED and TOS_WRB is for RAS_WRB. TOS_PRED and TOS_WRB are both initialized to zero. Transfer module II reads the stack entry indexed by TOS_PRED as predicted target of a return instruction or writes return address to RAS_PRED. Transfer module I only writes return address to RAS_WRB. Pointer module II and pointer module I are responsible for updating TOS_PRED and TOS_WRB, respectively.

According to the architecture given in Fig.1, the hardware cost of DSRAP can easily be compared to that of SSRAP. Suppose that the return address stack contains n entries of p bits each, and up to m branches are in-flight. A return address stack needs pn bits and a top-of-stack pointer needs log$n$ bits. So the total cost of DSRAP is 2*(pn+log$n$) bits, independent of the number of in-flight branches m. Single-stack return address predictors all require pn+log$n$ bits for the return address prediction structure, but additional bits for repairing are different. Saving the TOS pointer requires $m$log$n$ bits; saving the TOS pointer and TOS content requires m*(p+log$n$) bits; saving the entire stack requires m*(pn+log$n$) bits. Therefore, the total costs of these

three return address predictors are (pn+log$n$)+$m$log$n$, (pn+log$n$)+m*(p+log$n$) and (pn+log$n$)+m*(pn+log$n$), respectively, which are all dependent of m.

3.2 Design Issues

Stack operations. We employ the same stack model as described in [8]. RAS_PRED and RAS_WRB are all modeled as a circular LIFO buffer. A push operation causes the top-of-stack pointer to increase by 1 and a pop operation causes the top-of-stack pointer to decrease by 1. For the stack is circular, TOS will be (i+1) mod n after a push operation and (i-1) mod n after a pop operation, supposing that the stack has n entries and the current top-of-stack pointer is i.

RAS_PRED and RAS_WRB can both overflow and underflow. For RAS_PRED, an overflow occurs when a call is predicted taken and RAS_PRED is full, either because of stack corruption or because there are more calls in program than stack entries. The stack wraps around, which results in that the pushed return address overwrites the oldest stack entry. If an overflow has occurred in RAS_PRED, a later return that is predicted taken will cause an underflow by popping an already-popped entry, and as a result, the return instruction receives an invalid target. For RAS_WRB, an overflow occurs when a taken call is retired and RAS_WRB is full, only because there are more calls in program than stack entries. If an overflow has occurred in RAS_WRB, a later taken return will cause an underflow by popping an already-popped entry.

Stack updating. In the branch prediction stage, if a call instruction is detected and predicted taken, TOS_PRED increases by 1, and an associated return address is pushed onto RAS_PRED. If a return instruction is detected and predicted taken, the stack entry indexed by TOS_PRED pointer is popped from RAS_PRED as the predicted target address of the return instruction, and TOS_PRED decreases by 1. Calls and returns predicted not taken would not change RAS_PRED and TOS_PRED. Instructions except for calls and returns would not change RAS_PRED and TOS_PRED, either. RAS_WRB and TOS_WRB would not be changed in this stage.

In the writeback stage, if a taken call instruction is retired, TOS_WRB increases by 1, and an associated return address is pushed onto the RAS_WRB. If a taken return instruction is retired, TOS_WRB decreases by 1. Predicted target address of a return instruction is provided by RAS_PRED, not by RAS_WRB, so the data in the stack entry indexed by TOS_WRB pointer will not be used. Calls and returns not taken would not change RAS_WRB and TOS_WRB. Instructions except for calls and returns would not change RAS_WRB and TOS_WRB, either. If a branch misprediction is detected, all the entries in RAS_PRED are updated with the corresponding entries in RAS_WRB, and TOS_PRED is updated with TOS_WRB regardless of the reason of misprediction; otherwise RAS_PRED and TOS_PRED would not be changed in this stage.

4 Methodology

We use the modified version of SimpleScalar's sim-outorder [9] to collect results. SimpleScalar provides a simulation environment for modern out-of-order processors which allow executing speculatively. Table 1 shows the baseline configuration of the simulator used. The changes made to the simulator for our experiment are localized to the fetch stage and the writeback stage.

Table 1. Baseline configuration of the simulator

Parameter	Value
Processor core	
RUU size	64 instructions
LSQ size	40
Fetch queue size	8 instructions
Fetch/Decode width	4 instructions/cycle
Issue width	4 instructions/cycle(out-of-order)
Commit width	4 instructions/cycle(in order)
Functional units	4 integer ALUS, 1 integer multiply/divide, 1 FP add, 1 FP multiply
Branch prediction	
Branch predictor	Combining: 4K 2-bit selector, 12-bit history; 1K 3-bit local predictor, 10- bit history; 4K 2-bit global predictor, 12-bit history
BTB	2048-entry, 2-way
Mispredict penalty	2 cycles for misfetch, 7 cycles otherwise
Memory hierarchy	
L1D cache	64K, 2-way (LRU), 32B blocks, 1 cycle latency
L1I cache	64K, 2-way (LRU), 32B blocks, 1 cycle latency
L2	Unified, 8M, 4-way (LRU), 32B blocks, 12-cycle latency
Memory	100 cycles
TLBs	128 entry, fully associative, 30-cycle miss latency

Table 2. Benchmarks

Benchmark	Inputs	Warm Up Instructions
gzip	input.graphic	824M
vpr	ref inputs	105M
gcc	cccp.i	221M
mcf	inp.in	511M
crafty	crafty.in	926M
parser	ref.in	1051M
eon	ref inputs	26M
perlbmk	scrabble game	601M
gap	ref.in	271M
vortex	persons.1k	2451M
bzip2	input.compressed	2576M
twolf	ref inputs	255M

The SPEC2000 integer benchmarks [10] are used. All benchmarks are compiled using `gcc -O3 -funroll -loops`. The inputs of benchmarks are given in Table 2. Table 2 also shows the number of warm-up instructions performed to avoid the program's initial phases and any warm-up effects.

5 Simulation Results

From section 3 we notice that the hardware costs of DSRAP and SSRAP are different when the stack size is equal. So we will first evaluate DSRAP by comparing it with SSRAP from two aspects: when the stack size is equal and when the hardware cost is equal.

DSRAP can undo the effects of squashed instructions on the return address stack, so speculative execution would not cause return address mispredictions. Mispredictions in DSRAP are caused by unmatched call/return sequences or the stack overflow. The return address is unpredictable if there are unmatched call/return sequences. Fortunately, for most programs, unmatched call/return sequences are so rare that we can ignore the mispredictions caused by them. The overflow depends on the stack size only, and it can be avoided completely if the return address stack is large enough. Therefore, for DSRAP, we only need to consider the effects of stack size on prediction accuracy. In the second part of this section, we will show the effects of stack size on prediction accuracy of DSRAP.

5.1 Prediction Accuracy Evaluation

When the stack size is equal. Fig.2 shows the prediction accuracy of each return address predictor when the stack size is equal. The return address stack has 16 entries. Compared to saving the TOS pointer, DSRAP improves prediction accuracy dramatically, by an average of 9.58%. Compared to saving the TOS pointer and TOS content, DSRAP improves prediction accuracy remarkably, especially for *vpr*, *gcc*, *perlbmk*, *gap* and *twolf*, by 3.70%, 1.60%, 2.70%, 1.50% and 2.20% respectively. DSRAP can achieve the same prediction accuracy as saving the entire stack does, which accords with the theoretical analysis, because they can both undo the effects of squashed instructions on the return address stack.

When the hardware cost is equal. Fig.3 shows the prediction accuracy of each return address predictor when the cost is equal. The stack in DSRAP has 16 entries. The stack size cannot be too small; otherwise, stack overflow will dominate prediction accuracy. In order to consume the same hardware resources, the size of stack in saving the TOS pointer is fewer than 32, so does the size of stack in saving the TOS pointer and TOS content, and the size of stack in saving the entire stack is not more than 16 (m>1, generally). In our experiment, the size of stack in these three mechanisms is 32, 32 and 16 respectively, so the results will be biased towards these three kinds of return address predictors. Now with the same hardware resources consumed, compared to saving the TOS pointer, DSRAP improves prediction accuracy by 9.47%

on average, and compared to saving the TOS pointer and TOS content and saving the entire stack, DSRAP can also acquire higher hit rates.

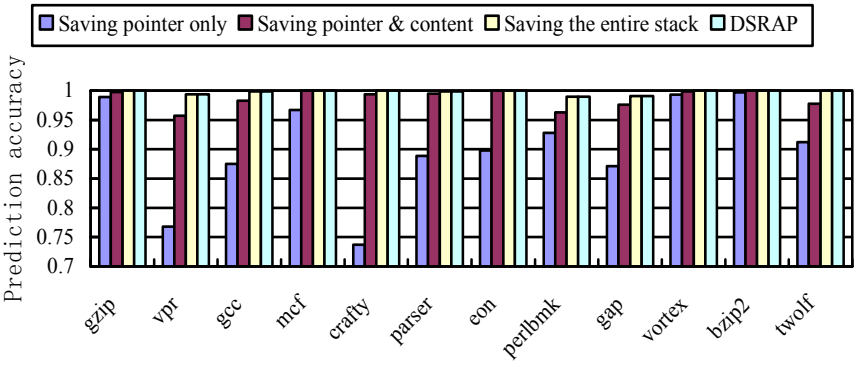


Fig. 2. Prediction accuracy of each return address predictor with a 16-entry stack

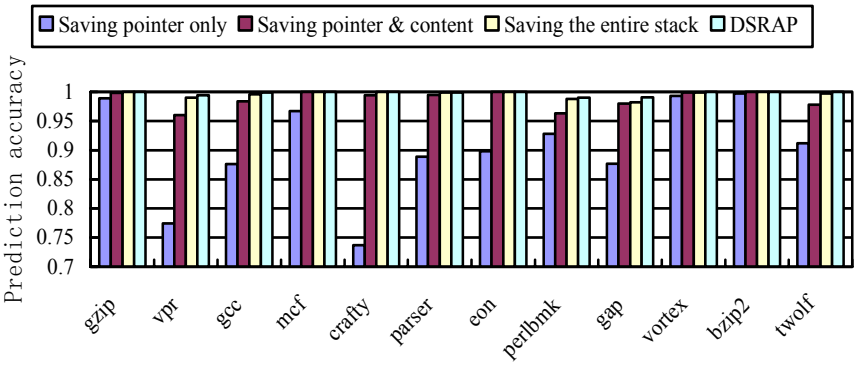


Fig. 3. Prediction accuracy of each return address predictor with the same cost

5.2 Effects of Stack Size on Prediction Accuracy

Fig.4 shows the prediction accuracy of DSRAP with different stack sizes. When the stack size is 0, BTB (Branch Target Buffer) provides the predicted target of a return instruction. For all benchmarks except for *gzip* and *bzip2*, prediction accuracy increases rapidly before the stack size reaches to 16. *vpr*, *gcc*, *perlbm*k, *gap*, *vortex* and *twolf* can benefit a little from a further increase to 16 entries. Almost no benchmarks can benefit from more entries than 16. So for most programs, a stack with 8-16 entries is enough to avoid stack overflow.

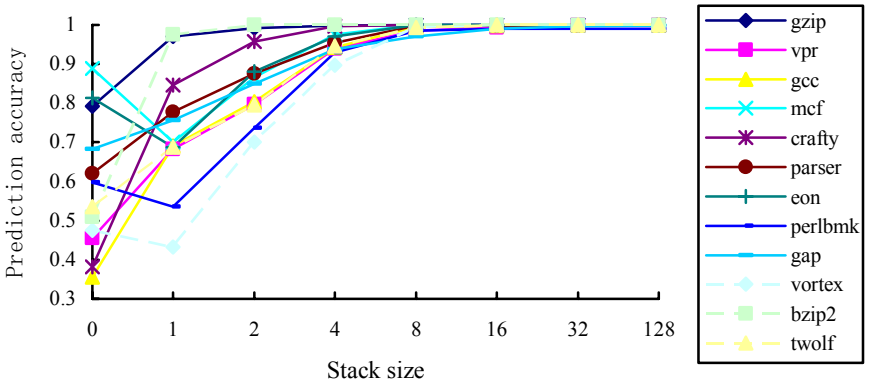


Fig. 4. Prediction accuracy of DSRAP with different stack sizes

6 Conclusion

Recall that the cost of each kind of SSRAP is linear with the number of branches in flight *m*. The larger *m* is, the higher the cost of SSRAP is. The cost of DSRAP, however, is independent of *m*. Therefore, with the pipeline of high-performance microprocessors deeper, DSRAP becomes cheaper relatively. Furthermore, DSRAP can achieve 100% return address prediction accuracy if mispredictions caused by the stack overflow or unmatched call/return sequences are ignored. Therefore, for the microprocessors with wide-issue and deep-pipeline, DSRAP is a good choice.

References

1. D.R.Kaeli and P.G.Emma: Branch history table prediction of moving target branches due to subroutine returns. In Proc. ISCA-18(1991)
2. C.F.Webb: Subroutine call/return stack. IBM Tech. Disc. Bulletin (1998)
3. Alpha 21164 Microprocessor: Hardware Reference Manual (1995)
4. L.Gwennap: Digital 21264 sets new standard. Microprocessor Report (1998)
5. H.Sharangpani, K.Arora: Itanium Processor Microarchitecture. IEEE Micro (2000)
6. C.McNairy, D.Soltis: Itanium2 Processor Microarchitecture. IEEE Computer Society (2003)
7. T.Yeh: Return address predictor that uses branch instructions to track a last valid return address. U.S. Patent No. 6,253,315(2001)
8. K. Skadron, P. Ahuja, M. Martonosi, D. Clark: Improving Prediction for Procedure Returns with Return address-Stack Repair Mechanisms. In Proceedings of the International Symposium on Microarchitecture (1998)
9. D. Burger, T.M. Austin, S. Bennett: Evaluating future microprocessors: the SimpleScalar tool set. TR-1308, Univ. of Wisconsin-Madison CS Dept. (1996)
10. The standard performance evaluation corporation. WWW site. http://www.specbench.org

Electronic Reading Pen: A DSP Based Portable Device for Offline OCR and Bi-linguistic Translation

Qing Wang[1,2], Sicong Yue[1], Rongchun Zhao[1], and David Feng[2]

[1] School of Computer Science and Engineering
Northwestern Polytechnical University, Xi'an 710072, P.R. China
qwang@nwpu.edu.cn
[2] School of Information Technologies
The University of Sydney, NSW2006, Australia
feng@it.usyd.edu.au

Abstract. In the paper, a portable off-line OCR and bi-linguistic translation system (Chinese to English, English to Chinese)—Electronic Reading Pen (ERPen) is designed and implemented. The constitution of ERPen hardware is designed and several modules, including CCD line array acquisition, wheel driven unit, FLASH management and USB interface are implemented. Moreover, the embedded software, consisting of image preprocessing, character segmentation and recognition, and corpus based postprocessing, is also discussed and implemented. A novel segmentation approach, central growth algorithm, is proposed and applied in ERPen system. Experimental results have shown that ERPen is effective to tackle printed character recognition and translation.

1 Introduction

With the rapid development of economy and broad communication between different regions and areas in the world, there exist a lot of language barriers in multi-language inter-translation for people so that a portable bi-linguistic or multi-linguistic translation device based on Optical Character Recognition (OCR) is widely demanded. In the last decades, printed character recognition [1,2] was gradually applied into commercial area from the experimental prototypes. A great number of products can be found, such as TH-OCR[3], HanOCR[4], FormAgent, DocAgent[5] and so on. However, most of the commercial systems are based on the document or form images scanned by the scanner and run on the desktop. Although a few products are designed and implemented in embedded systems, such as mobile phone, PDA, electronic dictionary and so on, the kernels of the character recognition are still based on online handwriting input instead of offline character recognition.

In the paper, an embedded off-line recognition and bi-linguistic interpretation system based on DSP chip, which is also called as Electronic Reading Pen (ERPen), is designed and implemented. The hardware framework of ERPen is

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 180–187, 2005.

designed and several key modules, including CCD line array acquisition, wheel driven unit, FLASH management and USB interface, are implemented. On the other hand, several key issues of the algorithms are discussed in detail, which include scanned image preprocessing, character segmentation and recognition, corpus based postprocessing, and translation. Meanwhile, a novel segmentation approach, central growth algorithm, is proposed in embedded software, which is based on region connectivity and dynamic programming. The cutting paths are obtained by central growth algorithm and a simplified cost function is defined to reduce the complexity of candidate decision. Moreover, a modified function based on second order derivative of the projection is also defined and used to evaluate the possibility of the cutting paths.

After a quick and easy scanning on the interested text area, the image of Chinese or English words is captured into ERPen system and corresponding text is translated and displayed on the LCD screen correctly. The ERPen system contains over 300,000 words and phrases in Chinese and English. The overall recognition rate for Chinese and English words are 99.2% and 98.3% respectively. At the same time, the processing time is very little. All of these results have proven that ERPen system is very effective and efficient.

The paper is organized as following. The framework of hardware system is described in the Section 2. In Section 3, the processing flow of the software system and key modules for segmentation, recognition and postprocessing are discussed. The experimental results and analysis are given in Section 4. Finally, the conclusion and further work are drawn in Section 5.

2 Hardware System

2.1 Constitution

The hardware system consists of a main module of DSP and four sub-systems, which are image acquisition, result display, USB interface, and storage management of feature set and dictionaries (Chinese to English — CTE and English to Chinese — ETC), as shown in Fig.1.

DSP Module. Considering that ERPen system has the requirements of real-time processing, low power consumption and small size of the shape, our initial design of processor unit is to make use of DSP chip since DSP chips have more advantages than traditional MCUs. In the final phase of the design, the kernel of DSP module utilizes TMS320-54X chip [6,7], on which image acquisition control, image processing, character segmentation and recognition, dictionaries management and word searching are carried out respectively.

Image Acquisition Module. The image acquisition part is mainly based on a 300 dpi Line Array CCD and wheel-driven control unit. The digital image from CCD is transferred into DSP module via parallel port and the sync signal and interrupts generated by wheel-driven control unit guarantee the image quality. Basically, the 8-bit gray scale image is captured and sent to binarization module. However, we found it very time-consuming so that a fixed threshold is used.

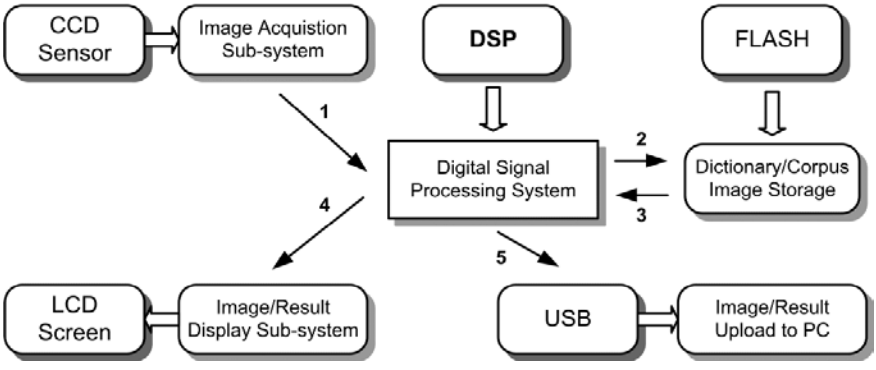


Fig. 1. The framework of the hardware modules and procedures in ERPen.

Result Display Module. The output sub-system is operated by DSP. The recognition and translation results are shown on the LCD (122 × 32 pixels). LCD is controlled by two micro chips, either of which is enabled by a trigger.

Storage Management Module. The storage module has 32M FLASH memory, which contains the extended programs, the labelled feature sets, ETC and CTE dictionaries, and some of scanned images, which can be upload to PC for further processing. DSP unit controls and accesses the data in FLASH through the data buses and extended I/O ports.

USB Interface Module. The scanned images can be uploaded into PC via USB interface. In ERPen, we select PDIUSBD12 chip as USB control unit, and 8 bit parallel data bus of PDIUSBD12 are connected with 8 bit data bus of DSP chip. USB interface is activated by PC and interrupted by DSP.

2.2 Improvements

In hardware design, we consider that digital line CCD array has digital output so that we omit the traditional scheme, which uses A/D modules and RAM to store the image. This strategy not only reduces the processing time, but also simplifies the hardware design. The image signal of CCD is controlled by three interrupts, which are Field Sync FST↔INT1, Line Sync LST↔INT2, and Pixel Sync PVB↔INT3. The following are these three interrupt functions.

```
interrupt void c_int1()
{
  cap=0;
  order=0;
}

interrupt void c_int2()
{
  order=0;
}
```

```
interrupt void c_int3()
{
  if (cap<32768) {
    if (order) {
      order=0; data2=CAPD;
      *(pStart+cap)=(data1&0xff)+(data2<<8);
      cap++;}
    else { order=1; data1=CAPD;}
  }
}
```

At the same time, the advantages of DSP chip are considered so that we can control and write/read FLASH memory directly by DSP. Moreover, USB interface is designed in the system in order to send scanned images or recognized results to PC and upgrade the firmware from PC, respectively. In some embedded system, USB protocol and interface may be implemented and controlled by MPU. Due to the requirements of ERPen, we utilize PDIUSBD12 and access it via TMS320-54X chip.

3 Software System

3.1 Framework

After we complete the hardware design, the software system is to solve the OCR and bi-linguistic translation. As a result, the software system is composed of the following modules: text region scanning, image preprocessing, character segmentation, character normalization and feature extraction, isolated character recognition, postprocessing, translation and display, as shown in Fig.2.

3.2 Key Issues and Implementation

Considering that DSP based hardware system has less resource and lower processing ability comparing with desktop computers, it is necessary to design high performance embedded software in order to satisfy the requirements of real time recognition and low power consumption. The key issues of software include character segmentation, feature extraction, classifier design and postprocessing.

Character Segmentation. Character segmentation is the first important step for OCR system so that there are many approaches proposed in the literatures, for example, vertical projection based method [2], knowledge based method [11], contour based method [12], foreground and background analysis based method [13] and so on. Besides the detection and segmentation of word strings, it is more difficult to separate the whole word into isolated characters. Since ERPen is designed to process Chinese and English words, it is necessary to deal with character separation properly. Due to the regularity of Chinese document and little connectivity among characters, it is comparatively easy to separate Chinese text lines by vertical projection. On the other hand, for English words, it is more

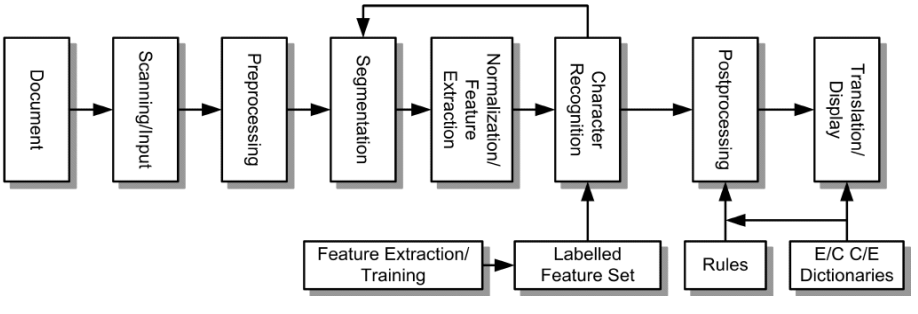


Fig. 2. The flowchart of the software modules in ERPen.

difficult to find out the gaps between characters since most of them may be so close as if they were connected regions, for example, 'ff', 'ft', 'fr', 'fi', 'rn', 'rt' and so on. On the other hand, connected components could be processed as split parts after image preprocessing, such as 'm'→'rn','w'→'vv' and 'd'→'cl' [10] in case of stroke pixel losing caused by the simplified preprocessing algorithm.

In the system, we propose a region connectivity and dynamic programming based approach to tackle character segmentation, which is also called as central growth algorithm (CGA). The detail of CGA is shown in the following steps.

1. Select the spare points on the center line as candidates for pre-cutting.

2. If there exists a path to approach the top and bottom of the word within a small strip from the candidate point, this path is regarded as a cutting path.

3. Otherwise, if the width of the cut region is greater than the predefined threshold or the average width of the character, it means that the box contains more than one character and these characters are touched with each other.

Herein, the vertical projection function $p(m)$, is utilized, and the second order derivative of $p(m)$, the Peak-Valley Projection Ratio Function $f(m)$, is also used to further cut the adhesive characters [9].

$$f(m) = [p(m-1) - 2 \times p(m) + p(m+1)]/p(m) \tag{1}$$

where $p(m)$ is the vertical projection value at m-th column.

4. Terminate if all of the candidates are processed completely.

Feature Extraction. After obtaining isolated characters, the coarse and fine statistic features are extracted from the normalized image (36×36). Suppose the stroke pixels are white (255) and the background pixels are black (0). Two kinds of features are used for classification, which are Coarse Periphery Feature (CPF) and Average Line Density (ALD).

1. **CPF** is based on the periphery image, which is generated by the run length of the blank line from the border to the first stroke pixel. CPF features reveal the shape information and periphery distribution of the characters. Before we extract CPF feature, we first obtain the peripheries of the character and fill out the inner vacancies of the borders, as shown in Fig.3b. Then the periphery image is partitioned into 4×4 grids, one of which has $9 \times 9 = 81$ pixels. Moreover, we re-partition the inner part (27×27) of the character into 3×3 meshes (shown in Fig.3d), and the number of white pixels is calculated. Therefore, we can get $16 + 9 = 25$ dimensional CPF features.

2. **ALD** is extracted from the first order differential image on vertical and horizontal directions respectively, which reflects the inner structure of the strokes. We first obtain differential images along the vertical and horizontal directions (see Eq.2 and 3). Then we divide the differential images into 9×9 grids and $81 + 81 = 162$ dimensional ALD feature can be extracted.

$$S_h(i,j) = |C(i+1,j) - C(i,j)| \quad i,j \in [1,N], C(N+1,j) = 0 \tag{2}$$

$$S_v(i,j) = |C(i,j+1) - C(i,j)| \quad i,j \in [1,N], C(i,N+1) = 0 \tag{3}$$

where $C(i,j)$ is the image intensity at pixel (i,j).

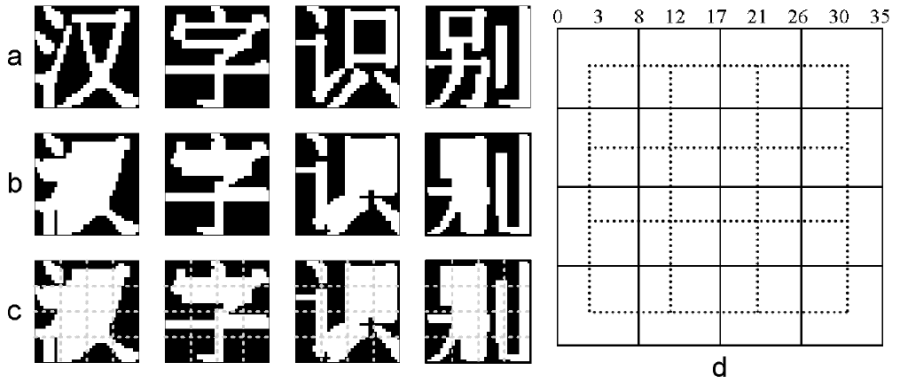


Fig. 3. Feature extraction and region partition. a). Original images; b). Vacancies-filled border images; c). Feature extraction; d). Partition scheme

Hierarchical Classifiers Design. In ERpen system, character classifiers are designed as three stages. The first one is applied as pre-classification so that 25 dimensional coarse features and City Block Distance (high tolerance to noise infection and low computation) are used to pre-select 30 candidates. Then the second stage classifier focuses on 162 dimensional fine features by Euclidean Distance in order to emphasize on the difference of similar characters and generates 10 fine candidates. At the last stage, the similarity measure is computed (see Eq.4) since it is more precise than distance measures.

$$R(X,G) = \sum_{i=1}^{m}(x_i \times g_i) \Big/ \sqrt{\sum_{i=1}^{m} x_i^2 \times \sum_{i=1}^{m} g_i^2} \tag{4}$$

where $X = (x_1, x_2, \ldots, x_m)$ is the feature vector of the unknown character and $G = (g_1, g_2, \ldots, g_m)$ is the template vector from training samples, respectively.

Post-processing. Basically, the correctly recognized words are translated into English or Chinese by CTE or ETC dictionaries directly. However, since there exist wrong cases more or less in accordance with segmentation and recognition errors, it is necessary to design a serial of rules to modify the recognized word. From the experiments, we find out that four kinds of errors for English words were caused by wrong segmentation and recognition, which are, 1) Replacement error: recognize one character as another one; 2) Mergence error: recognize several characters as one character; 3) Deletion error: miss characters in recognition phase characters, and 4) Insertion error: insert redundant characters into a word.

To deal with issues mentioned above, many methods using word based or semantic level post-processing are proposed and can produce good performance [14]. However, we can not simply carry out these algorithms in ERPen system since all of them have high computation complexity. After a great number of experiments on database with 70,000 words, we have summarized a serial of rules, including replacement, insertion and deletion, to solve these problems.

4 Experimental Results

4.1 Recognition Rate

The recognition of Chinese and English characters by ERPen is listed in Table.1. The upper part of the table shows the correct recognition rates by three stage classifiers for Chinese character. The most right column is the final recognition result after using corpus and rules based post-processing. The lower part of the table shows the corresponding results for English characters.

Since we have fully considered the effectiveness of the character segmentation and recognition, the experimental results are reasonable. The precise results show that the algorithms of text image preprocessing, character segmentation and recognition and post-processing are effective and satisfied with the requirements of ERPen system.

4.2 Processing Time

Besides high recognition rate and correct translation result, the computation time is also important when evaluating the performance of ERPen system. Through a lot of experiments, we get the average computing time, which is 0.153s for each Chinese word and 0.287s for English word, respectively.

The performance of speedy processing is based on the simplification of the preprocessing procedure and some other algorithms. The results of processing times have shown that the design and implementation of hardware system is efficient and will fulfill the demand of real time processing.

4.3 Problems

Although we have successfully designed the hardware and embedded software, the problem of high consumption of power still exists. According to the original design, ERPen can work using 2 AAA Alkaline batteries and last for one month continuously. However, the developed system can not satisfy this requirement.

5 Conclusions

In the paper, we describe the hardware and software frameworks and implementation of ERPen, a kind of portable device using offline printed character

Table 1. Recognition rate for printed Chinese and English characters

	Classification method	Correct Recognition Rates (%)	Correct rates after postprocessing(%)
Chinese	first stage	89.7	
	second stage	99.0	
	similarity	99.1	99.2
English	first stage	83.4	
	second stage	94.8	
	similarity	95.4	98.3

recognition and bi-linguistic translation. We fully consider the characteristics of TMS320-54x chip and the requirements of real time processing: small size, and huge storage in terms of hardware design. At the same time, an embedded software configuration for ERPen is well designed to deal with several key processes of image preprocessing, character segmentation and recognition, and rules based postprocessing. Experimental results have shown that ERPen has high recognition and correct translation rate (99.2% for Chinese words and 98.3% for English words) and high processing speed. In the future work, we will put our emphasis on the problem of power consumption.

Acknowledgements

The work described in the paper was partially supported by National Natural Science Fund (No. 60403008), "The Developing Program for Outstanding Persons" fund by Northwestern Polytechnical University, Natural Science Foundation of Shaanxi Province, P. R. China, and ARC grant, Australia.

References

1. Mantas, J.: An overview of character recognition methodologies. Pattern Recognition. **19** (6), (1986) 425–430
2. Casey, G., Lecolinet, E.: A Survey of methods and strategies in character segmentation. IEEE Trans Patt. Anal. Mach. Inetll. **18**(7), (1996) 690–706
3. http://www.wintone.com.cn
4. http://www.hw99.com/
5. http://www.ceresoft.com
6. Zhang, X., Cao, T.: Principle and development application of DSP chips (Second Edition). Publish House of Electronics Industry, China. (2000)
7. Dai, M., Zhou, J.: Structure, principle, and application of TMS32054X DSP. Press of BUAA, China. (2001)
8. Khan, S.: Character segmentation heuristics for check amount verification. Ph.D Dissertation, MIT, USA (1998)
9. Liu, G., Wei, F. etal: A segmentation method of cursive handwritten digit string based on limited dynamic programming. J. of Beijing Univ. of Post and Telecommunication. **26**(1), (2003)14–18
10. Lu, Y., Haist, B. et al: An accurate and efficient system for segmenting machine-printed text, 5^{th} Advanced Technology Conf. U.S. Postal Service. **3**(1992), 93–105
11. Liu, G., Ding, X. etal:Knowledge synthesis decision based character segmentation algorithm. Computer Engineering and Application. **17**(2002)59–63
12. Strathy, N., Suen, C., Krzyzak, A.: Segmentation of handwritten digits using contour features. Proc. 2^{nd} Int. Conf. on Document Analysis and Recognition(ICDAR), Tsukuba Japan. (1993) 577–580
13. Chen, Y., Wang, J.: Segmentation of single- or multiple-touching handwritten numeral string using background and foreground analysis. IEEE Trans Patt. Anal. Mach. Inetll. **22**(11), (2000)1304–1317
14. Dey, S.: Adding feedback to improve segmentation and recognition of handwritten numerals. Ph.D Dissertation, MIT, USA (1999)

Formal Co-verification for SoC Design with Colored Petri Net*

Jinyu Zhan, Nan Sang, and Guangze Xiong

School of Computer Science & Engineering
University of Electronic Science & Technology of China
Chengdu 610054, China
zhanjy@uestc.edu.cn

Abstract. The complexity of SoC is increasing rapidly. It is an important trend that SoC design is always based on the reuse of both IP cores and software components. In consequence, new verification techniques are needed, which overcome the limitations of traditional methods and are suitable for SoC at the same time. This paper introduces a computational model for SoC based on colored Petri net, formulates the IP cores, components and user defined logics, and presents a method to translate the architecture design into the colored Petri net model. And a formal co-verification approach of SoC using CPN tools is also proposed. The method concentrates on verifying the correctness of the design. An example of the audio and video architecture design of the PDA platform illustrates the effectiveness of our approach on practical applications. Finally, the experimental results are given.

1 Introduction

With the development of embedded system technology, the integration level of hardware is higher and higher. SoC (System on Chip) becomes a mainstream design approach of embedded systems. And the development of SoC is a challenge to the verification of embedded systems.

For the levels of complexity typical to modern electronic systems, traditional validation techniques like simulation and testing are neither sufficient nor viable to verify their correctness. First, these techniques may cover just a small fraction of the system behavior. Second, long simulation times and bugs found late in prototyping phases have a negative impact on time-to-market.

Co-verification is a new technology to verify the software and hardware of embedded systems. Different from traditional method, co-verification emphasizes parallel processing and the interaction between software and hardware. Co-verification is not only beneficial to reduce the time-to-market and design cost for the embedded products, but also gives a better understanding of the system behavior, contributes to uncover ambiguities, and reveals new insights of the systems.

* This work was supported by the National High Technology Research and Development Program (863), under Grant No. 2003AA1Z2210.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 188–195, 2005.

One main idea of SoC is the reuse of the IP (Intellectual Property) cores, which can reduce the time-to-market and design cost [1]. The conception of software components is like IP cores of SoC. Both of them emphasize the importance of design reuse. This paper presents a hardware/software formal verification method for the SoC design based on both IP cores and software components. Once the architecture of the IP cores, components and UDLs(user defined logics) are defined, the correctness of the whole system can be verified.

This paper is organized as follows. The related works are given in section 2. In section 3, we formally define the IP cores, components, UDLs and the whole systems using colored Petri nets. In section 4, we illustrate our method through an example. In section 5, our approach is proved effective by verifying the audio and video architecture design of a practical PDA platform. Finally, some conclusions are drawn in section 6.

2 Related Works

In the field of hardware/software co-verification of the embedded systems, there are two research directions. One is simulation, and the other is formal verification. In the simulation method, the hardware of the whole embedded system is specified in hardware description language such as Verilog HDL, VHDL, SystemC, HandleC, and is established in software. It can realize the integrated debugging and modify the faults of hardware freely, which avoids the waste of manpower and material resources. But with many restrictions, the hardware description language can only describe a subset of the hardware of common embedded systems. Therefore, more and more scientists turn their attention to formal verification. Edmund M. Clarke presented the theory of Model Checking [2] based on finite automata for hardware formal verification. In recent years, Model Checking has been extensively used in hardware/software co-verification of embedded systems. Predicate Abstraction presented by Graf and Saidi [3], which is a method to verify the correctness of software, is modified by many scientists and used in hardware/software co-verification of embedded systems in these years. Luis Alejandro Cortes presented PRES method [4][5], which uses Petri Net to describe hardware and software of the embedded systems at first and then verifies the description in hybrid automata model which was presented by Alur [6]. But these methods mentioned above seldom consider the problems about the verification of SoC based on IP cores and components, which is the purpose of this paper.

3 Formal Representation

IP cores are reused modules in SoC, while software components are reused modules in the software systems. Though they belong to hardware subsystems and software subsystems respectively, both of them embody the idea of module reuse. Therefore, they can be formulated into one model. In this paper, reused modules are used to represent both IP cores and components.

The verification of SoC consists of three parts, the verification of the reused modules, the UDLs and their interconnections. Reused modules, which are provided by core vendors, must be verified before delivery, and can be regarded correct. There are

many techniques to test or verify the UDLs [7][8] [9][10][11]. The correctness of the UDLs can be guaranteed through these methods. Therefore, the verification of SoC puts emphasis on verifying the interconnections among IP cores, components and UDLs.

Both UDLs and reused modules can be regarded as black boxes. Once the inputs are given, the outputs are defined. So they can be represented as "if $input_1$, $input_2$, ..., $input_m$ then $output_1$, $output_2$, ..., $output_n$", in which $input_1$, $input_2$, ..., $input_m$ are the inputs of the UDLs or the reused modules, and $output_1$, $output_2$, ..., $output_n$ are the outputs of them. They all can be formulated into Petri net model shown in Figure 1.

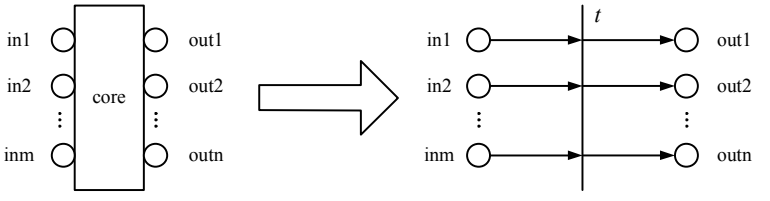


Fig. 1. Formulate a core into a Petri net model

Definition 1. *SoC based on IP cores and components can be modeled in a nine-tuple CPN(Colored Petri Net)= $(\Sigma, P, T, A, N, C, G, E, S)$, where*

(1) Σ is the color set, determines the types, operations and functions that can be used in the net. It is assumed that the color sets have at least one element each;

(2) $P = \{p_1, p_2, ..., p_m\}$ is a finite non-empty set of places, represents the inputs and outputs of the cores in SoC;

(3) $T = \{t_1, t_2, ..., t_n\}$ is a finite non-empty set of transitions, represents the cores in SoC;

(4) A is a finite non-empty set of arcs such that, $P \cap T = P \cap A = T \cap A = \Phi$, defines the flow relation between places and transitions; $I \subseteq P \times T$ is a finite non-empty set of input arcs, which defines the flow relation from places to transitions; $O \subseteq T \times P$ is a finite non-empty set of output arcs, which defines the flow relation from transitions to places;

(5) N is a node function, defined from A into $P \times T \cup T \times P$, which maps each arc into a tuple where the first element is the source node and the second element is the destination node;

(6) C is a color function, defined from P into Σ, which means that C maps the place p to a color set;

(7) G is a boolean expressions, called guard function, which maps the transition t to the Boolean function and are needed to evaluate to "true" in-order when the transition fires;

(8) E is an arc expression function, which maps the arc a to an expression, and a transition in a CPN is enabled if it is possible to bind the variables in such a way that the arc-expressions of all the input arcs evaluate when tokens are present at the corresponding input places;

(9) S is the initialization function, which specifies the initial state of the Petri net.

Definition 2. *A Marking* $M : P \rightarrow \{0, 1\}$ *is a function that denotes the absence or presence of tokens in the places of the net. Therefore, the Petri net* CPN *is safe or 1-bounded, that is, a place may hold at most one token for a certain marking.* $M(p) = 1$ *whenever the place* p *is marked, otherwise* $M(p) = 0$.

4 Verification and Analysis

In this section, a verification method for SoC's correctness is represented, using the model introduced above, is presented. It is illustrated through an example shown in Figure 2. According to the definitions in Section 3, the Petri net model equivalent to the architecture design of the example in Figure 2 is given in Figure 3.

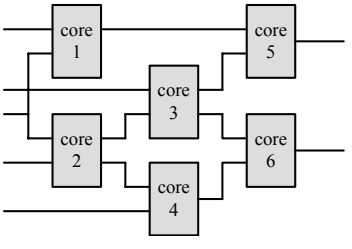


Fig. 2. An example of an architecture design

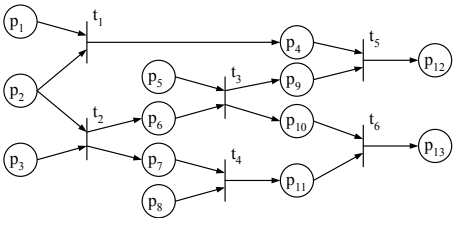


Fig. 3. The Petri net model of the example

Therefore, the correctness of the architecture design is transformed into the analysis of Petri net model. There are two main types of analysis that can be performed on SoC architecture design represented in Petri net. The first one is liveness analysis. A given marking, i.e. absence or presence of tokens in places of the net, may represent the state of the system in the dynamic behavior of the Petri net. It is very important whether the system ends at the needed states or is dead in the dynamic process. Therefore, deadlock is a very important problem. Second is reachability analysis. In the dynamic behavior of the Petri net, the designer could be interested in proving that the system eventually reaches a certain state whose marking represents the completion of a task.

There are two traditional methods to analyze the Petri nets. One is the reachability tree, and the other involves the matrix equations. But both are manual and very complex. Therefore, we use CPN Tools [12] to analyze the Petri net model of the architecture design. CPN Tools is a tool for editing, simulating and automatically analyzing Colored Petri nets. All the needed analysis results of the Petri net model can be obtained through CPN Tools. Then we can get the state graph of the example shown in Figure 4 from CPN Tools. CPN Tools generates an analysis report of the Petri net model shown in Table 1 which contains statistical information, reachable information and liveness information about the state graph. Therefore, we can conclude that the architecture design of the example is not correct from Figure 4 and Table 1.

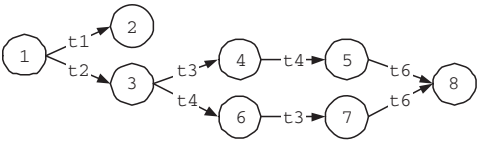


Fig. 4. The state graph of the example

Table 1. Analysis results of the example

State Graph	Reachable Properties	Liveness Properties
Nodes: 8	Reachable State: $p_1, p_2, p_3, p_4, p_5, p_6,$	Dead Transition: t_5
Arcs: 8	$p_7, p_8, p_9, p_{10}, p_{11}, p_{13}$	Live Transition: t_1, t_2, t_3, t_4, t_6
Status: Not Full	Unreachable State: p_{12}	

Now let us analyze the fault of the architecture design. p_2 is the input place of the transition t_1 and transition t_2, while the CPN is 1-bounded according to definition 2. So there are exclusive relations between transition t_1 and transition t_2. We modify the architecture design shown in Figure 5 to get rid of the exclusive relations. The modified Petri net model is given in Figure 6. Then we can get the state graph of the net shown in Figure 7 from CPN Tools. The analysis results of the modified Petri net model are shown in Table 2. Then we can get that the modified architecture design is correct from the data.

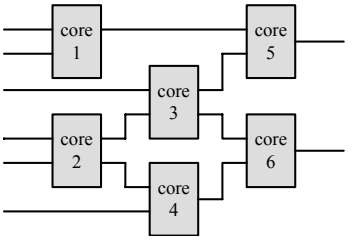


Fig. 5. the modified architecture design

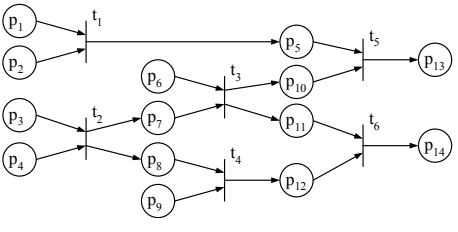


Fig. 6. The modified Petri net model

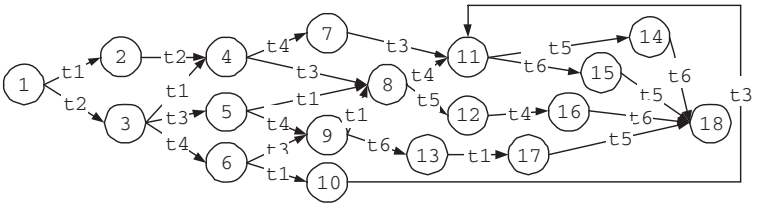


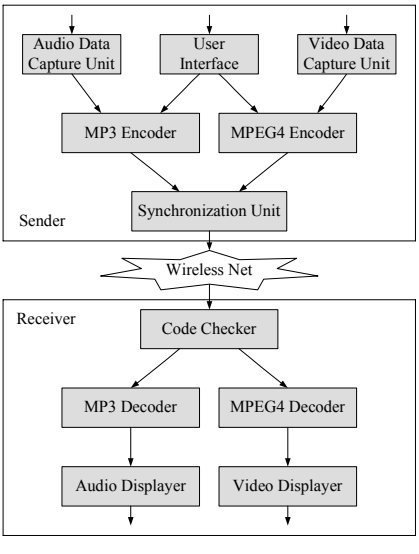
Fig. 7. The state graph of the modified architecture design

Table 2. Analysis results of the modified architecture design

State Graph	Reachable Properties	Liveness Properties
Nodes: 18	Reachable State: All	Dead Transition: None
Arcs: 26	Unreachable State: None	Live Transition: All
Status: Full		

5 Verification of a Practical System

In this section, we will illustrate the verification of a practical system using our approach. The video and audio architecture design of a PDA platform is shown in Figure 8. The Petri net model equivalent to the architecture design is given in Figure 9. And we can get the state graph of the Petri net model shown in Figure 10 and the analysis results shown in Table 3 from CPN Tools. Therefore, the audio and video architecture design of the PDA platform is correct according to Figure 10 and Table 3.



Audio Data Capture Unit and Video Data Capture Unit are IP cores. They transform voice and image into audio and video information.

User Interface is a user define logic. It deals with user requirements and sends them to MPEG4 and MP3 Encoder.

MP3 Encoder and MPEG4 Encoder are IP cores. They deal with the compression and coding of the video and audio information.

Synchronization Unit is a component. It deals with the synchronization problems of the audio and video, and sends information from sender to receiver through the wireless net.

Code Checker is a component. It analyzes and checks the information from sender, and outputs audio and video information.

MP3 Decoder and MPEG4 Decoder are IP cores. They deal with the decompression and decoding of the audio and video information.

Audio Displayer and Video Displayer are hardware devices. They display the voice and image.

Fig. 8. The audio and video architecture design of the PDA platform

6 Conclusions and Future Work

This paper presents a methodology to perform formal co-verification of SoC design, which is composed of IP cores in hardware design, components in software design and UDLs. A colored Petri-net-based design representation is used to capture important features of SoC. The co-verification technique based on architecture design with

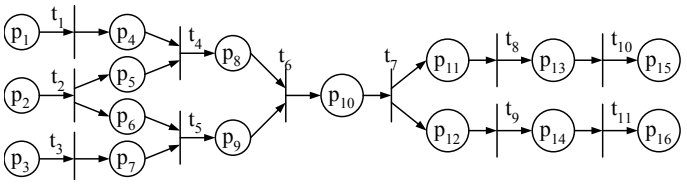


Fig. 9. The Petri net model of the audio and video architecture design

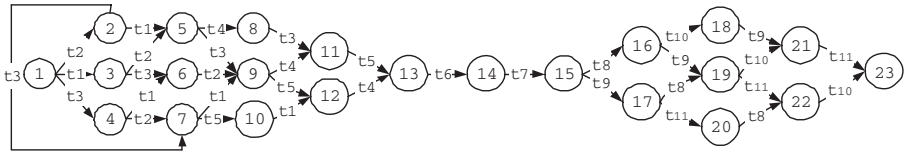


Fig. 10. The state graph of the audio and video architecture design

Table 3. Analysis results of the audio and video architecture design of the PDA platform

State Graph	Reachable Properties	Liveness Properties
Nodes: 23	Reachable State: All	Dead Transition: None
Arcs: 34	Unreachable State: None	Live Transition: All
Status: Full		

IP core-component reuse smoothly integrates with communication. The method can translate the architecture design of SoC into a colored Petri net model simply and intuitively. The colored Petri net model can be automatically analyzed many properties such as liveness, reachability by CPN Tools. We also illustrate the verification of the audio and video architecture design of the PDA platform using our approach, and give the experimental results.

The performance, such as timing requirement and power consumption, is very important in SoC design. In this paper, we concentrate on the correctness of the architecture design, but don't consider the timing requirement and power consumption. Therefore, these are the problems worth for further research.

References

1. Haase, J.: Design methodology for ip providers. In: Proc. DATE 1999. (1999) 728–732
2. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. (1999)
3. Graf, S., Saidi, H.: Construction of abstraction state graphs with pvs. In: CAV'97. (1997) 72–83
4. A, C.L., P, E., Z, P.: Formal coverification of embedded systems using model checking. In: The 26th Euromicro Conference. (2000) 106–113
5. A, C.L., P, E., Z, P.: Verification of embedded systems using petri net based representation. In: The 13th International Symposium on System Synthesis. (2000) 149–155

6. Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transaction on Software Engineering **22** (1996) 181–201
7. Marinissen, E., Arendsen, R., Bos, G.: A structured and scalable mechanism for test access to embedded reusable cores. In: The International Test Conference. (1998) 284–293
8. Yoneda, T., Fujiwara, H.: A dft method for core-based systems-on-a-chip based on consecutive testability. In: The 10th Asian Test Symposium. (2001) 193–198
9. Iyengar, V., Chakrabarty, K., Marinissen, E.J.: Test wrapper and test access mechanism co-optimization for system-on-chip. Journal of Electronic Testing: Theory and Applications **18** (2002) 213–230
10. T, S., Y, Y., T, H.: Between-core vector overlapping for test cost reduction in core testing. In: The 12th Asian Test Symposium. (2003) 268–273
11. O, S., A, O.: Parity-based output compaction for core-based socs [logic testing]. In: The 8th IEEE European Test Workshop. (2003) 15–20
12. Online: CPN Tools, (http://wiki.daimi.au.dk/cpntools/)

Hardware for Modular Exponentiation Suitable for Smart Cards

Luiza de Macedo Mourelle[1] and Nadia Nedjah[2]

[1]Department of Systems Engineering and Computation
[2]Department of Electronics Engineering and Telecommunications
Faculty of Engineering
State University of Rio de Janeiro, Brazil
{ldmm, nadia}@eng.uerj.br

Abstract. Smart cards use integrated circuits instead of magnetic tape. Its architecture includes a processor, memory, input/output and, possibly, a cryptographic coprocessor. The cost of smart cards is directly related to the size of the integrated circuit. Our present focus is the cryptographic coprocessor, which usually uses a public-key cryptosystem. In these cryptosystems, the main operation is the modular exponentiation, which is performed using successive modular multiplications. This operation is time consuming for large operands, which is always the case in cryptography. Here, performance is another matter of concern. For fast software or hardware cryptosystems, one needs thus to reduce the total number of modular multiplications required. In this paper, we propose a fast and compact hardware for computing modular exponentiation using the *m*-ary methods, applying the addition chain method for the pre-processing step. The cryptographic hardware is low-cost and concise, offering a good solution for smart cards.

1 Introduction

The RSA encryption scheme [1], [2] is an example of public-key cryptographic system. This kind of cryptosystem often involve raising large elements of some groups fields, such as GF(2^n) or elliptic curves [3], to large powers. The main operation in such cryptosystems is the modular exponentiation, which is performed by successive modular multiplications. As the plain text of a message or the cipher text are usually large, i.e. 1024 bits or more, it is essential to attempt to minimise the number of modular multiplications performed, in order to improve time requirements of the encryption/decryption operations. Hardware architecture implementations of the RSA cryptosystem are widely studied [4].

The paper-and-pencil method to compute $C = T^E \bmod M$ requires $E-1$ modular multiplications, computing all powers of T: $T \to T^2 \to T^3 \to ... \to T^{E-1} \to T^E$. The window-based methods [1] consist of algorithms that perform modular exponentiation with a nearly minimal number of modular multiplications. These methods have a pre-processing step, which can be optimised with the use of the addition chain methods.

Smart cards are plastic cards with an integrated circuit on it. They guarantee more security than the plastic cards with magnetic strip do. The smart cards are more

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 196-202, 2005.

sophisticated, which contain a processor, memory, input/output components and a cryptographic coprocessor. A typical architecture for smart cards is depicted in Fig 1, in which the processor is based on the 8051 or 6805 microcontrollers. Smart cards cost about ten times more than common plastic cards, with magnetic strip, do. The cost of smart cards is proportional to the area of the integrated circuit, which is directly associated to the complexity of the operations required and the architecture of the system. Therefore, in order to minimise the cost of smart cards we should target at the area required for its implementation. One of our aims is to obtain a cryptographic coprocessor that offers minimum area with optimal performance.

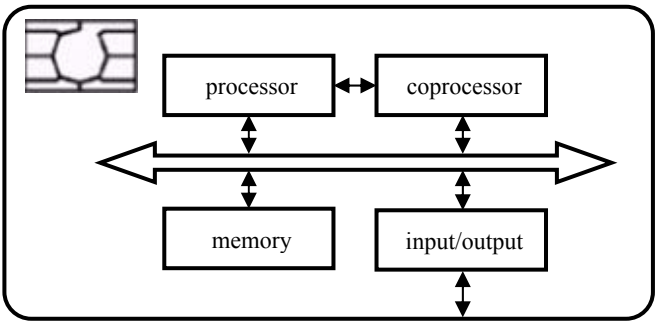


Fig. 1. Smart cards architecture

This paper is organised as follows: first, we concentrate on describing the *m*-ary methods; then, we introduce the addition chain method; next, we present the Montgomery's algorithm used for the modular multiplication; thereafter, we propose the architecture design for the *m*-ary modular exponentiation, based on the addition chain method, together with an iterative and low-cost hardware for Montgomery's algorithm; finally, we summarise the work presented throughout this paper and draw some conclusions.

2 Modular Exponentiation Based on the M-ary Methods

The *m*-ary methods for modular exponentiation [1] consist of three major steps: *(i)* partitioning the binary representation of the exponent E in l-bit windows; *(ii)* pre-computing all possible powers in windows one by one; *(iii)* iterating the squaring of the partial result l times to shift it over, and then multiplying it by the power in the next window, if this is different from 0. In other words, the *m*-ary methods partition E in p windows of length $l = \log_2 m$, where m is a power of two. Algorithm 1 describes the *m*-ary algorithm, wherein M and E represent the modulus and exponent of the cryptosystem, T and C stand for the text and the ciphertext, respectively, and, finally, V_i denotes the decimal value of the ith window, where $p\text{-}1 < i < 0$.

```
Algorithm 1. MME(T, M, E)
1:  Partition E into p l-bit windows;
2:  for i = 2 to m-1 Compute T^i mod M;
3:  C := T^(V_p-1) mod M;
4:  for i := p-2 downto 0 do
5:     C := C^(2^l) mod M;
6:     if V_i ≠ 0 then C := C×T^(V_i) mod M;
7:  return C;
end.
```

The pre-processing step, shown in lines 2 and 3 of Algorithm 1, calculates all possible powers of T, according to the window size l. However, we do not know which powers will effectively take part in the final computation. In order to reduce the amount of computation performed in the pre-processing step, we use the addition chains method to obtain a sequence of powers to yield T^E.

3 Addition Chains Method

An *addition chain* of length r for a positive integer N is a list of positive integers (a_0, a_1, a_2, ..., a_r) such that $a_0 = 1$, $a_r = N$ and $a_k = a_i + a_j$, $0 \le i \le j < k \le r$. Finding a minimal addition chain for a given positive integer is an *NP*-hard problem. It is clear that a short addition chain for exponent E gives a fast algorithm to compute $T^E \bmod M$ as we have $T^{a_k} = T^{a_i} \times T^{a_j}$, if $a_k = a_i + a_j$.

A generalisation of the concept of addition chain is that of addition sequence. An *addition sequence* for a list of positive integers V_1, V_2, …, V_p, such that $V_1 < V_2 < \ldots < V_p$, is an addition chain for integer V_p, which includes V_1, V_2, …, V_p. The length of an addition sequence is the number of integers that constitute the chain. An addition sequence for a list of positive integers V_1, V_2, …, V_p will be denoted by $\mathcal{S}(V_1, V_2, \ldots, V_p)$. For instance, considering V_1=3, V_2=7 and V_3=11, a possible addition sequence would be (1,2,3,4,7,9,11). An addition sequence of minimal length (or simply minimal addition sequence), for the values of the partitions included in the non-redundant ordered list $\wp(E)$, would optimise the number of modular multiplications required in the pre-processing step of the m-ary methods for computing $T^E \bmod M$ (line 2 of Algorithm 1). Finding this minimal addition sequence is an *NP*-hard problem and we use genetic algorithms to solve it [5].

4 Hardware Architecture

Fig. 2 introduces the hardware architecture for the m-ary methods. The modular multiplications are performed using Montgomery's algorithm [6], for which we have developed a hardware architecture (MODMULT) described later on in this paper. The pre-processing step computes the powers of T, based on the addition sequence

provided, and stores these powers in a local memory (MEM). Each position of this memory consists of two kinds of information: the high-order bits store the exponent, as provided by the addition sequence, and the low-order bits store the corresponding power of *T*. At the beginning, position 0 contains de decimal value 1 in its high-order bits and *T* in its low-order bits.

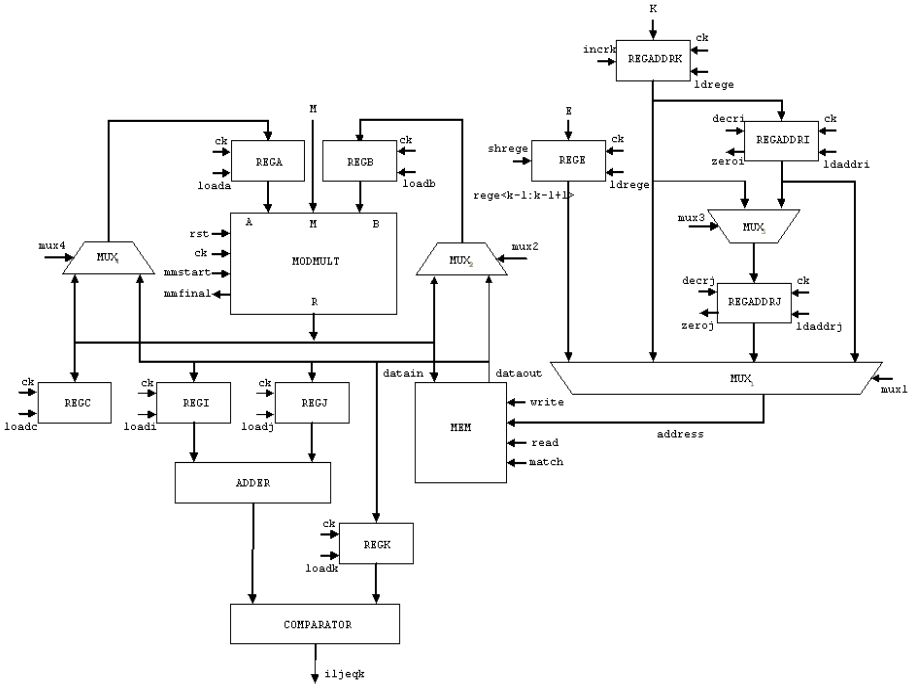


Fig. 2. The architecture of the m-ary hardware

The powers are computed according to the addition chain rule: $a_K = a_I + a_J$, $0 \leq I \leq J < K$. *K* is initialised to 2 (REGADDRK=2), *I* to 1 (REGADDRI=1) and *J* to 1 (REGADDRJ=1). The memory location addressed by *K* is read and the high-order bits of the word are loaded in REGK. The memory location addressed by *I* and *J* are, subsequently, read and the word loaded in REGI (high-order bits) and REGA (low-order bits), and in REGJ (high-order bits) and REGB (low-order bits), respectively. The sum of REGI and REGJ is, then, compared to REGK. Once valid *I* and *J* are found, the current contents of REGA and REGB are, then, modular multiplied. The result is stored in memory location *K*. This value is, then, incremented and *I* and *J* are initialised to *K*-1. The search for other valid values begins, by successively decrementing *I* and *J*, until found. For example, using the initial values declared, the first power computed is T^2.

Each partition of the exponent *E* will be used to address the memory to obtain the corresponding pre-computed power of *T*, as defined in line 3 of Algorithm 1. The local memory is, in fact, an associative memory (MEM), in order to read the data based on the value of the current exponent partition. During this step, signal *match* is asserted for every read cycle from MEM.

In each iteration of the exponentiation step, the partial result C is raised to the 2^l power and, then, multiplied by T^{V_i} modulo M, when V_i is not a zero partition (see lines 5 and 6 of Algorithm 1). The values of T^{V_i} modulo M are obtained from the associative memory, according to the current partition of the exponent E. In order to obtain the value of the current partition, we store exponent E in shift register REGE, from which the most significant partition is retrieved to address the associative memory (see line 3 and 6 of Algorithm 1). When a new partition is required, register REGE is left-shifted l times. REGL is loaded with the length of the partition (l) and decremented for each shift operation performed. The square-and-multiply loop (starting in line 4 of Algorithm 1) consists of two main phases:

1. The first one performs l squaring of the partial result. For this purpose, the partial result is fed-back to inputs A and B of the modular multiplier of Fig. 2;
2. The second phase performs the modular multiplication of the partial result with the pre-computed power of T, when the current partition is not zero. The power of T, i.e. T^{V_i} modulo M, is read from the associative memory, at the location matching the most significant partition of register REGE.

The square-and-multiply loop is executed until the least significant partition of E is reached. REGP is loaded with the number of partitions (P) and decremented once in each iteration. The final result is then loaded into register REGC.

The pre-processing step consists of performing $r-1$ modular multiplications, while the exponentiation step consists of $l(p-1)+q \mid 0 \leq q \leq p-1$ modular multiplications, where q is number of non-zero partitions. The operands, however, differ from one multiplication to another. The main work of the controller consists of setting up the right operands for each one of these modular multiplications. The controller interface signals are set according to the synchronous finite state machine $SM=(S_0, Q=\{S_0, S_1, \ldots, S_{19}\}, F=\{S_{19}\}, \delta)$, wherein S_0 is the initial state, Q is the state set, F is the set of final states and δ is the state transition function, presented as follows:

```
S0: Initialise the system; If start = 1 Then go to S1;
S1: REGE <= E; REGADDRK <= K;
S2: Read MEM(REGADDRK); REGADDRI <= K; REGADDRJ <= K;
S3: REGK <= MEM(REGADDRK); Decrement REGADDRI and REGADDRJ;
S4: Read MEM(REGADDRI);
S5: REGI <= high-order bits of MEM(REGADDRI);
    REGA <= low-order bits of MEM(REGADDRI);
S6: Read MEM(REGADDRJ);
S7: REGJ <= high-order bits of MEM(REGADDRJ);
    REGB <= high-order bits of MEM(REGADDRJ);
S8: If (REGI + REGJ) = REGK Then go to S12;
    Else if REGADDRJ=1 Then go to S10;
S9: Decrement REGADDRJ;
S10: Decrement REGADDRI;
S11: REGADDRJ <= REGADDRI;
S12: Start the modular multiplier;
S13: If modular multiplier has finished Then go to S14;
```

```
S14: Stop the modular multiplier;
S15: Write the result in MEM(REGADDRK);
S16: Increment REGADDRK; If REGADDRK > addition chain size Then go to S2;
S17: Read MEM(most significant window of REGE); Decrement REGP;
S18: If modular multiplication finished Then go to S19;
S19: REGA <= MEM(most significant window of REGE);
     REGB <= MEM(most significant window of REGE);
     If there are no more squaring to do Then Go to S21;
S20: If modular multiplication finished Then go to S19;
S21: Decrement REGP; REGL <= partition size;
S22: Decrement REGL; left shift REGE;
     If there are no more bits to shift Then go to S23;
S23: If the new partition is not zero Then go to S24
     Else If there are more partitions Then go to S20 Else go to S26;
S24: REGB <= MEM(most significant partition of REGE);
S25: If the modular multiplier finished Then
     If there are more partitions Then Go to S27;
S26: REGA <= modular multiplication result;
     REGB <= modular multiplication result into REGB; Go to S20;
S27: Indicate end of operation; If start signal unasserted Then Go to S0;
```

5 Results

The design is specified in VHDL [7] and a functional simulation [8] is then performed. The project is synthesised [8] and the area and time requirements are registered for different parameters. The figures are listed in Table 1. The hardware area is given in CLBs while the response time is given in nanoseconds (ns). The results show clearly that the proposed hardware is very efficient: it requires very much less hardware area and encrypts/decrypts very much faster.

Table 1. Area and time requirements for the *m*-ary hardware that uses a minimal addition sequence vs. the *m*-ary hardware that does not

Operand size	*m*	*M*-ary Hardware with addition sequence		*M*-ary Hardware without addition sequence	
		area (CLBs)	*time* (ns)	*area* (CLBs)	*time* (ns)
64	2	492	3.1	509	17.3
	4	441	2.9	897	15.0
128	2	811	7.3	912	22.9
	4	721	5.8	1777	20.1

6 Conclusions

In this paper, we have presented a fast and compact hardware implementation for the modular exponentiation based on the *m*-ary methods, using a minimal addition

sequence of exponents. As the window size *l*, related to the partitioning of the exponent *E*, increases, so does the amount of possible powers of *T* to compute during the pre-processing step. Therefore, instead of computing all the possible powers of *T* in the pre-processing step of the *m*-ary methods, we compute only those powers present in the minimal addition sequence. We use genetic algorithms to obtain this sequence. The pre-processing time decreases using a minimal addition sequence of exponents, as the window size *l* increases. During the iterative step of the *m*-ary methods for the exponentiation process, the pre-computed powers are retrieved from the memory. Hence, the memory size required is smaller than if we compute all the possible powers, thus reducing the overall area.

Acknowledgements

The authors wish to acknowledge the financial support provided by Fundação de Amparo à Pesquisa no Estado do Rio de Janeiro (FAPERJ) and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for the development of this research.

References

1. Koç, Ç.K., *High-speed RSA Implementation*, Technical report, RSA Laboratories, Redwood City, California, USA (1994).
2. Rivest, R.L., Shamir, A. and Adleman, L., *A method for obtaining digital signature and public-key cryptosystems*, Communication of ACM (1978), vol. 21, no.2, 120-126.
3. Menezes, A.J., *Elliptic curve public key cryptosystems*, Kluwer Academic (1993).
4. Eldridge, S.E. and Walter, C.D., *Hardware Implementation of Montgomery's Modular Multiplication Algorithm*, IEEE Transactions on Computers (1993), 42(6), 619-624.
5. Nedjah, N., Mourelle, L.M., *Efficient Pre-processing for Large Window-based Modular Exponentiation using Genetic Algorithms*, Proceedings of the 16th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, Loughborough, England (2003), LNAI 2718, 625-635.
6. Montgomery, P.L., *Modular Multiplication without Trial Division*, Mathematics of Computation (1985), vol. 44, 519-521.
7. Navabi, Z., *VHDL - Analysis and Modeling of Digital Systems*, McGraw Hill, Second Edition (1998).
8. Xilinx Inc., *ISE 6.1i*, http://www.xilinx.com.

PN-based Formal Modeling and Verification for ASIP Architecture[1]

Yun Zhu, Xi Li, Yu-chang Gong, and Zhi-gang Wang

Dept. of Computer Science, University of Science and Technology of China, Hefei, Anhui 230026, China
yukiyun@mail.ustc.edu.cn

Abstract. This paper presents a novel extended timed Petri Net model called PNPM — Petri Net based Representation for Pipeline Modeling, and a verification scheme based on PNPM called PPL-MC — PNPM and Lambda Calculus based Model Checker They focus on formal modeling and verification especially for ASIP architecture with pipeline structure. In this paper, PNPM elements have been defined, and their validity and usage are demonstrated. Also, the scheme of PPL-MC is introduced.

1 Introduction

Figure 1 shows the design flow [1] of ASIP (Application Specific Instruction Processors). Architecture DSE (Design Space Exploration) is an important step. To uncover the flaws earlier and to realize the automatization of DSE need a validation mechanism. Formal verification [4] is a widely-used method for validation, though it is underdeveloped. Many are concentrated on some layers below architecture or certain part of architecture, such as [2, 3].

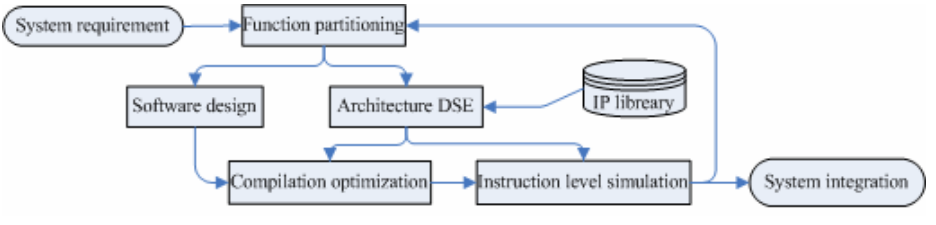


Fig. 1. Flow of the software-hardware co-design

We want to do formal verification for our ASIP architecture DSE, so the first thing come to us is how to formally model the ASIP architecture. Petri Net [6] is a graphical and mathematical modeling tool especially applicable to asynchronous and

[1]Supported by the National Natural Science Foundation of China under Grant No.60273042; the Natural Science Foundation of Anhui Province of China under Grant No.03042101.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 203-209, 2005.

concurrent system. It [7, 8] is netlike, and can be represented as a 3-tuple N=(P, T; F) with certain restrictions. We call P and T in the triple *place-set* and *transition-set* of N respectively; and *arc-set* F reflects their relationships. *Token*s are added to express status messages and their distribution is called *marking*. The dynamic behavior of PN model is guarded with its *firing rule*. Implementation of a transition must be atomic. For $\forall x \in P \bigcup T$, the input and output of x are called *pre-set* and *post-set* of x (denoted by ${}^{\bullet}x$ and $x^{\bullet}$) respectively.

Petri Net is a precise system, and can be extended according to certain requirements. The application of Petri Net for performance analysis of networks and concurrent systems is already well developed. But applications to architecture modeling are relatively few. The basic Petri Net is too simple to describe the structure and behavior of complicated architecture, and the size of PN model is a big problem.

Aimed at the design of ASIP architecture, this paper first presents a novel extended timed Petri Net model called PNPM (Petri Net based representation for Pipeline Modeling), which can describe target architecture in a succinct and precision way. Then a formal verification method based on PNPM is introduced, taking advantage of PNPM, PPL-MC can efficiently verify various aspects of the target system..

2 PNPM — An Extended Timed Petri Net Model

2.1 Pipelined Architecture and PNPM

Pipeline is a key technique in modern ASIP architectures. It brings high performance as well as some technical problems. Various pipelines are exploited in different domain, but the basis of pipeline is always its units, behavior, timing and the relationships among them.

There are two kinds of units in pipeline: storage units and execution units. Denoting them by places and transitions respectively seems rational. PNPM is based on *1-PN* since the pipeline registers always have mono-value. It extends a token to a pair $k = (v, r)$, where v is the *token value* implying register value, and r is the *token time*. So the marking M for each place p, when it has token in it, $M(p)$ stands for the token pair in it. $M(p).v$ denotes the token value, $M(p).r$ denotes the token time. v_p and r_p are brief notations.

We also make each transition T have a formal *description* which can be stored and used as independent text file. Shown as below, it includes information about action, timing, and signal control and so on.

```
description_T6{                          /*description for transition T6*/
    pipelinestage = 3 ;                  /*EXE */
    enable = (v_p9 = "R" ) ;             /*P9 connects toT6 with a testing arc*/
    action ={
            v_p6 = v_p3 op v_p4;         /*system behavior of T6*/
            }
    timedelay= 6 ;                       /*firing time for T6*/
    }
```

Enabling of transition in PNPM depends on the token pair in the place with testing arc connecting to it. So the *enabling* and *firing* are defined as follows:

Definition 2-1. Transition $t \in T$ is **enabled** at marking M iff:

$$(\forall p \in {}^{\bullet}t : M(p) \neq \varepsilon) \wedge (\forall p \in t^{\bullet} : M(p) = \varepsilon) \wedge description\_t.enable .$$

Definition 2-2. The **firing** of an enabled transition $t \in T$ changes a marking M into a new marking M', denoted by $M[t > M'$. As a result of firing the transition t, the following events occur:

$$\begin{cases} \forall p \in {}^{\bullet}t, M(p) = \varepsilon & (1) \\ \forall p \in t^{\bullet}, M({}^{\bullet}t).v \xrightarrow{description\_t.action} M'(p).v & (2) \\ \forall p \in t^{\bullet}, M'(p).r = \max\{M({}^{\bullet}t).r\} + description\_t.timedelay & (3) \end{cases}$$

2.2 Modification for PNPM

For modeling the ASIP architectures with PNPM more precisely and succinctly, we need to do some modification in PNPM.

Transition should not be disabled when its output places have tokens because register with value can still get a new value to replace the old one. Structure with inhibitor arc and testing arc shown in figure 2 (b) is a solution to it. *T1* in figure 2 (c) is an encapsulated symbol of *T1* and *T1'*. And transitions in PNPM all imply such.

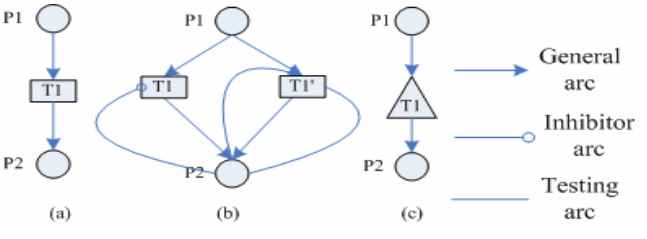


Fig. 2.transition with value-replacement semantics

Certain storages like system *memory* and *register file*, which have many memory cells, are modeled in PNPM as *multi-place*, which encapsulate many normal places into one abstract place. When simulating or evaluating, needed cells are parsed out from it. We denote multi-place by a ring as in figure 3.

Thus *manual-transition* is a correspondence to *multi-place*. *T5* denoted by ▭ *in figure 3* is a *manual-transition* express *instruction decoding*. It is coincided with normal transition in the form, and could be implemented as a sub-PNAM; i.e., PNPM supports systems modeled at different levels of granularity with transitions representing simple arithmetic operations or complex algorithms. Also, we use *transient* (a thick bar) to simply express transition with token replication semantics in 0 time delay. In Petri Net, one token can only take part in one transition at a time. By using *transient T1* in figure 3, the value of token in *P1* (PC) can now be used by both *T2* (*fetching* instruction) and *T3* (new PC counting) simultaneously.

2.3 Formal Definition of PNPM

Since all the extended PNPM elements can be constructed from the basic PNPM elements, we introduce the formal definition of PNPM.

Definition 2-3 A PNPM system is a 7-tuple, $\Sigma = (P, T; F, B, C, A_T, M_0)$ where:

(1) $(P, T; F)$ is a net;

(2) $B: T \to Boolean$. $\forall t \in T$, $B(t)$ is a Boolean value or a Boolean expression about the token values of tokens in the places connected to t with testing arcs;

(3) $C: T \to \Omega$. Ω is a finite set of sentences. $\forall t \in T, C(t) \in \Omega$, sentence in C (t) is either a evaluation of the token values in ${}^\bullet t$ with the token values in $t^\bullet$, or a behavior description of them after a "*stall*" instruction;

(4) $A_T: T \to (N, B(T), C(T), R^+ \cup \{0\})$. $\forall t \in T$, $A_T(t)$ is the value of the 4-tuple (*pipelinestage,enable,action,timedelay*) corresponding to "description _t", where N is natural numbers, and $R^+ \cup \{0\}$ is the set of non-negative real numbers;

(5) $M: P \to (\Psi, R^+ \cup \{0\}) \bigcup \varepsilon$. $\forall p \in P$, $M(p)$ denotes the token pair associated with the place p. Ψ can be any possible type of the token value, and token time is a non-negative real number. $M(p) = \varepsilon$ means that no token in p at marking M. M_0 is the initial marking of the net.

3 A PNPM Model for a 5-Stage Pipeline

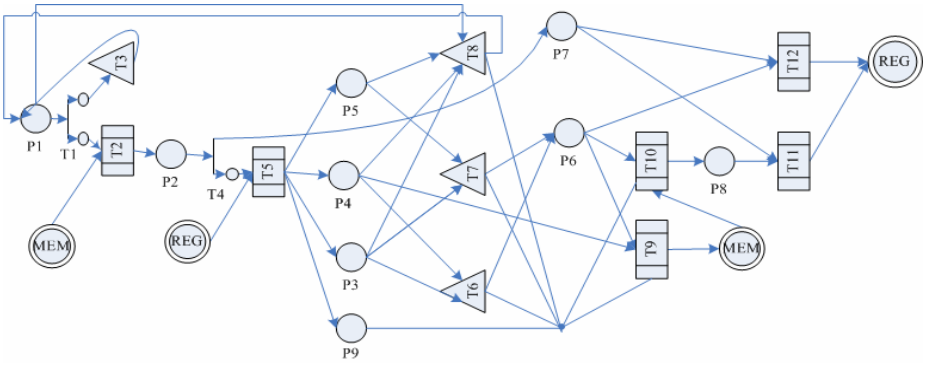


Fig. 3. A PNPM Model for a 5 stage pipeline

Figure 3 is a graphical representation of a PNPM model. It is a five-stage pipeline with stages *IF*, *ID*, *EXE*, *MEM*, *WB* in turn. The associations between PNPM elements and architecture elements are shown in Table 1. The token in P9 indicates the control signal with the value *L*, *S*, *R* or *B* corresponding to the Instruction *Load*, *Store*, *Reckon* and *Branch* in turn. The token in P9 is issued from *T5* in stage *ID*, and the place *P9* is connected to *T6*, *T7*, *T8*, *T9* and *T10* with testing arcs.

T1 and *T4* are *transient*s. *T2*, *T5*, *T9*, *T10*, *T11* and *T12* are *manual-transition*s connected to *multi-place*s. Descriptions of all these transitions in figure 3 can refer to *description_T6* in 3.1.

Table 1. Correspondence between place/transition and architecture element

Places	Arch. Units		Transitions	Architecture Behaviors
P1	PC		T2	IF/ID.IR = MEM [PC]
P2	IF/ID.IR		T3	PC = PC + 4
P3	ID/EXE.A		T5	ID/EXE.A = REG [IF/ID.$IR_{6...10}$] ID/EXE.B = REG [IF/ID.$IR_{11...15}$] ID/EXE.Imm= $(IR_{16})^{16}$##REG .$IR_{16...31}$
P4	ID/EXE.B		T6	EXE/MEM.ALUoutput = ID/EXE.A op ID/EXE.B
P5	ID/EXE.Imm		T7	EXE/MEM.ALUoutput = ID/EXE.A+D/EXE.Imm
P6	EXE/MEM. ALUoutput		T8	if (ID/EXE.A = ID/EXE.B) PC = PC + ID/EXE.Imm – 4 else PC = PC
P7	EXE/MEM.B		T9	MEM [EXE/MEM.ALUoutput] = ID/EXE.B
P8	MEM/WB.LMD		T10	MEM/WB.LMD= MEM [EXE/MEM.ALUoutput]
P9	(ID.signal)		T11	REG[IF/ID.$IR_{11...15}$] = MEM/WB.LMD
			T12	REG [IF/ID.$IR_{16...20}$] = EXE/MEM.ALUoutput

4 PNPM Based Formal Verification

A formal verification can simply be expressed as "*M* |= *P*". Where "*M*" is a system modeling like state machine, "*P*" is a set of required properties usually represented by logics. "|=" is the technique to deal with the relationships between "*M*" and "*P*".

Based on the PNPM, which is designed for the formal modeling for ASIP architectures, we have constructed a formal verification scheme. As shown in figure 4, PPL-MC (PNPM and Lambda Calculus based Model Checker) uses a Petri Net based model PNPM to represent the target system, and Lambda Calculus[5] to specify the expected system properties. The Lambda Calculus (LC) is a formal system designed to investigate function definition, function application and recursion. It can be used to cleanly define what a "computable function" is. Any computable function can be expressed and evaluated using LC formalism with a single transformation rule (variable substitution) and a single function definition scheme. It is thus equivalent to turing machines, and system behavior restrictions can be easily described dependent on its expressiveness and canonicity.

Since we use PNPM as "*M*" and use LC as "*P*" in our formal verification. The most pivotal point left is the verification process "|=", viz. the PPL-MC scheme. See figure 4, it is composed of five main modules: two for input pretreatment, two major checking modules, and an output management module.

The "*PN-based Checking*" module uses Petri net technique to check the initial PNPM models on their reachability, liveness, boundness, fairness, and so on. The "*L-Formula Preliminary*" module unrolls and simplifies the input LC formulas to regular and straightforward form. If there exists illegality of the input, a feedback through the export will return to the beginning.

PPL-MC can perform two kinds of verification, one is "*Equivalence Checking*" between two PNPM models, and the other is "*Property Satisfying*" between a PNPM model and L-formulas. To spread the whole system model in fine grit will inevitably lead to state space exploration, and the design of the complex system becomes infeasible. In virtue of the modularized and hierarchical modeling ability of PNPM, there can be a series of PNPM model of a single system at different levels and different pipeline stages. So the consistencies between them become very important. We provide a mechanism based on *net morphism* theory to automatically check the equivalence of them. Property satisfying is realized by PPL-MC with an interpreter. It constructs mappings between atomic transitions with formalized atomic operation in PNPM and the atomic functions in LC; and the satisfaction between M and P can be based on them. Also, the checking can be performed globally or in a particular stage. Moreover, since PNPM has timing ability, constraints with timing can also be checked.

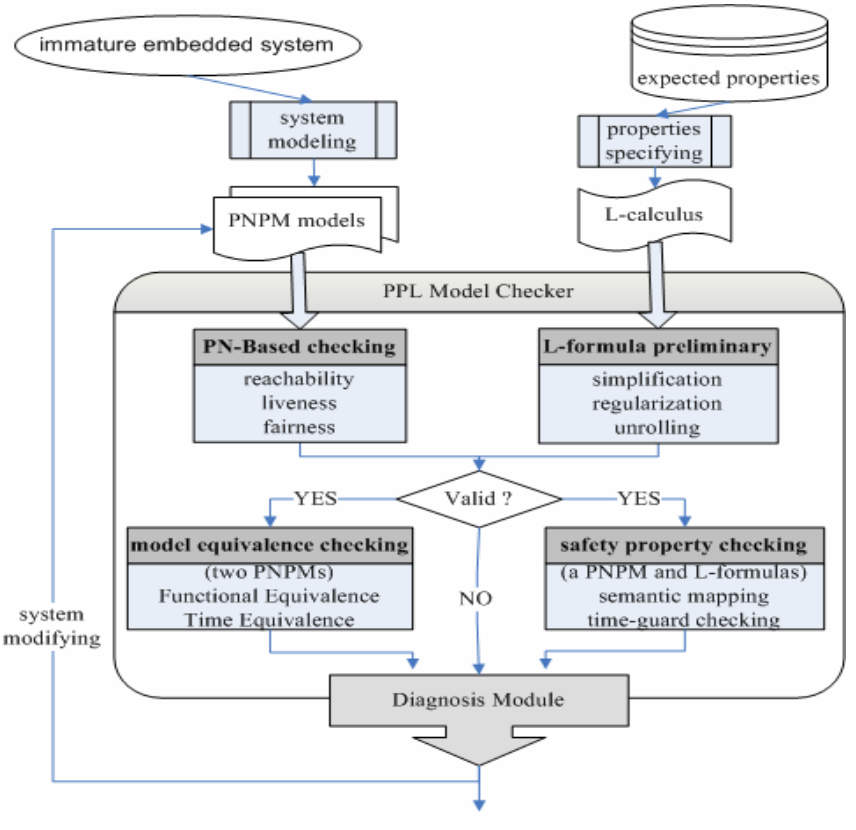


Fig. 4. The PPL-MC Scheme

All these checking results above are drawn to a "*diagnosis module*", which automatically analyzes and decides what to do next, that is, to feedback and make modifying, or to proceed.

5 Conclusions

We have introduced and formally defined PNPM, a Petri Net based novel model aimed at formal verification in DSE. The model is simple, intuitive, and applicable to any abstract level of architecture in DSE. PNPM is a graphical and mathematical model with extensions to capture important characteristics of various architectures by describing the units, behavior, timing and their relationships structural and hierarchical. We have presented an encapsulation approach to improve the correctness and succinctness of PNPM. In addition, a five-stage pipeline has been studied to illustrate the applicability of our approach to practical systems.

We have also constructed the scheme of a PNPM and Lambda Calculus based Model Checker called PPL-MC. It uses PNPM as underlying sustainment for system modeling. Detailed design of its modules is ongoing. Aimed at formal verification of ASIP Architecture, taking advantage of PNPM, PPL-MC can efficiently verify various aspects of the target system.

References

1. Li X, Zhou XH, Xiong Y, Lu L, Zhao ZX. XP-ADL: A key issue in Software and Hardware Codesign[A].DPCS2002[C]. Wuhan: 2002. 100-104.
2. R.S.Tupuri, J.A.Abraham. A Novel Functional Test Generation Method for Processors using Commercial ATPG[A]. Proceedings Intl Test Conference[C]. San Jose, California, 1997. 743-752.
3. Jeffrey Su, David Dill, Jens Skakkeb. Formally Verifying Data and Control with Weak Reachability Invariants[A]. FMCAD'98[C]. California, USA: Phillip J. Windley, 1998-11-04. 387-402.
4. E. M. Clarke, J. M. Wing. *Formal methods: state of the art and future directions*. ACM Computing Surveys, 28(4):626--643, Dec. 1996.
5. H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of Studies in Logic and the Foundations of Mathematics. NorthHolland, Amsterdam, The Netherlands, Revised Edition, 1984.
6. Yuan CY. Principle of Petri Net[M]. Electronics Industry Press, 1998 (in Chinese). 25-75.
7. Jiang CJ. Behavior Theory and Applications of Petri Net[M]. Higher Education Press, 2002. 19-28.
8. Lin C. Stochastic Petri Net and System performance evaluation[M]. Tsinghua University Press, 1999. 1-44.

The Design and Performance Analysis of Embedded Parallel Multiprocessing System

Guanghui Liu, Fei Xia, Xuejun Yang, Haifang Zhou, Heng Zhao, and Yu Deng

Institute of Computer, National University of Defense Technology
410073 Changsha, China
nudtlgh@sohu.com

Abstract. The performance of traditional embedded satellite-carried system is limited by various conditions. It makes people find out a better solution. This paper presents a solution of embedded parallel multi-processing system which orients to satellite-carried application. The system consists of two double-processor parallel systems. In addition, theory analysis and program simulation are used to analyze the system's performance. Compared with single-processor embedded system, the design can not only improve system performance observably but also satisfy the requirement of satellite business processing better.

1 Introduction

The embedded system is used more and more widely. However, in some severe circumstances which are restricted by various factors (e.g. temperature range, radiation-hard ability), its processing ability cannot be largely improved. Some application fields demand for high processing ability of embedded system (e.g. the image data processing of aerospace remote sensing, multimedia data processing), making the single processor of embedded system unqualified for the job. This forces system designers to explore new methods to improve the processing ability of embedded system. A reasonable solution for overcoming the low performance of embedded application is the parallel multi-processing technology. This paper discusses the design and performance analysis of embedded multi-processing system based on satellite-carried parallel computer system and furthermore explores on the concept of embedded multi-processing system. Presently, the most representative project about satellite-carried computer system is SCS750 of Maxwell Technology, Besides, Proton 100k™ of Space Micro Inc and ISC series of GD-AIS Company are all good candidates for satellite-carried computer. All the above solutions employ the parallel computer architecture, they provide enough processing ability for all kinds of satellite business management and pay load processing which is increasing steadily.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 210-215, 2005.

2 System Design

The architecture of the highly-reliable embedded multi-processors system for satellite and its behavior are described in detail in this section.

2.1 System Architecture

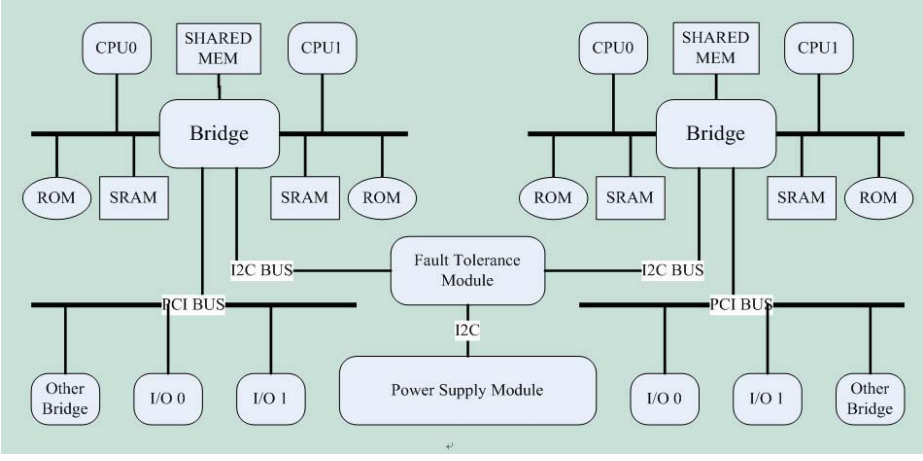


Fig. 1. The architecture of Highly-reliable Embedded Multiprocessors system

The solution is designed for satellite-carried system, whose working environment is critical. Therefore it demands high reliability. The whole system exploits multi-level reliable design strategies, including dual-computer cold backup in system level, dual-CPU processing and degraded running in unit level and redundant design in logic level.

The whole system is consisted of CPU module, fault tolerance module, I/O module and power supply module. As can be seen in figure.1, the CPU module, bridge, and the I/O module coupled with them make up of an independent computer. There are two computers of this kind in this satellite-carried system, which are named computer A and computer B (Computer B is the cold backup for computer A) respectively. CPU modules are consisted of two CPUs, two local memories, two ROMs, one shared-memory and one bridge.

On one hand, the fault tolerance module monitors the working state of system by connecting to bridge through I^2C bus. On the other hand, it is connected to power module by I^2C bus, achieving the function of degraded running on a certain computer (A or B) or system switching when the fatal fault is detected.

When the system is powered on, the power supply module supplies voltages for computer A in default and the whole system begins to work. The fault tolerance module monitors the working state of computer A. When there is an error coming from a certain CPU, system enters the state of degraded running. Meanwhile, with the assistance of the normal-stated CPU of computer A, the fault tolerance module boots computer B and completes the state transition of the whole system. Then computer B

begins to work. The process above can be called system switching. If there is an error coming from a certain CPU of the current computer (computer B), the system enters the state of degraded running again, without taking the system switching process. In this situation, the normal-stated CPU can still assure of completing all system functions.

As the key unit in CPU module, bridge includes several important functions:

- Protocol transformation between local memory bus and PCI bus, which transforms one of the two types of transaction on two buses to the other,
- Bus control and arbitration, which sequences the concurrent I/O accesses from two CPUs and implements the arbitration function for PCI bus,
- Synchronized communication mechanism between two CPUs, which assures the shared-memory of being accessed critically, and thus assures the data in shared-memory of consistency,
- DMA and interrupt control, which deal with the DMA request and interrupt request from peripheral equipments,
- Redundant interface for I^2C bus, which makes the communication between bridge and fault tolerance module reliable.

The architecture of the solution introduced here is in some sense similar to SMP, but each processor has its own local SRAM. The memory access model is likely between NORMA and UMA. CPUs for this solution are all commercial processors and communications between them are not through crossbar network but bridge. The shared-memory is hung onto the bridge and the whole system looks symmetric. All processors can access the shared-memory and I/O devices coequally but each processor has its own operating system located in the local SRAM. Because processors used in space is much slower than which on the ground, data can be accessed directly from or to local SRAM despite of the processor-memory gap without the help of cache. Therefore, as shared data is located in shared-memory and private data in local memory, problem of cache-coherence can be avoided.

2.2 Strategies of Improving Performance

The system uses close coupling architecture of shared-memory. Here gives the strategies of improving performance in this design:

- Exploiting dual-CPU strategy, which obviously makes the processing capability of system more powerful,
- System is symmetric and shared-memory is introduced, so high degree of parallelism can be exploited,
- The two CPUs each have their local memory for private data, which brings two major benefits. First, it is a cost-effective way to scale the memory bandwidth if most of the accesses are to the local memory in the node. Second, it reduces the latency for accesses to the local memory,
- The two CPUs share one memory space, which is employed for data exchange and transfer. This communication mechanism has several advantages compared to message-passing mechanism. First, it makes programming easier when the communication patterns among processors are complex or vary dynamically during

execution, and similar advantages simplify compiler design. Second, when developing applications using the familiar shared-memory model, it focuses attention only on those accesses that are performance-critical. Third, it aims at the characteristics of relative independence among applications and small communication items. The overhead for communication will be lower, use of bandwidth will be more efficient, and efficiency of communication will be higher.

3 Parallel Performance Analysis

The index of performance to judge the parallel processing capacity of systems is application speedup. Speedup can be presented as $S= T_s/T_p$, "T_s" represents serial execution time, "T_p" represents parallel execution time. How to exploit the parallel processing capacity to the utmost is related extremely to the software and hardware structure of the system and the characteristics of application. The design employs the architecture of dual-computer redundant cold backup. On the normal state, two CPUs of single computer are running collaterally. They employ shared-memory to communicate with each other and their program space is independent.

The software system of multi-processors on satellite is an embedded real-time operating system. The kernel code of system software can be controlled less than 100kB. If instructions are loaded from ROM directly, the program will not use dynamic storage space. Because of banding with applications, the system cost can be ignored, so the speedup is extremely decided by task allocation and exploiting of the parallelism degree of application programs. However, the measurement of accurate speedup must be combined with particular application type. When tasks are distributed and the proportion of communication is controlled reasonably, the speedup can approach linear acceleration.

Analysis as far as application, there are two main parallel application patterns: coarse grained task level parallelism suitable to the management of tasks on satellite, and fine-grained algorithm level parallelism suitable to pay load. This plan can support both parallel patterns above.

3.1 Coarse-Grained Parallelism

By analyzing the applications on satellite, it is known that tasks on satellite are not calculation-intensive but independent with each other and the grains of communication is also small. So the execution pattern of the program on the dual-CPU system should be coarse-grained. What's more, in order to simplify design and satisfy the requirement of real-time, the task distribution on the dual-processor is initialized statically, and dispatched dynamically. It can be supposed that within a long execution time of full load, n tasks are equally distributed to 2 CPUs. Define the tasks' number on the two CPUs differently as "n1" and "n2", and the serial execution time of the task No. k is "t_s^k", then the single CPU's execution time of n tasks $T_s = t_s^1 + t_s^2 + \ldots + t_s^n$, and the parallel time $T_p = \max\{ T_{n1}, T_{n2}\}$, T_{n1}, T_{n2} represent the execution time on each CPU. Suppose $T_p = T_{n1} \geq T_{n2}$, $T_{n1} = t_s + t_c$, among these, $t_s = (t_s^1 + t_s^2 + \ldots + t_s^{n1}) \geq T_s/2$, which is serial execution time of n1 tasks and "t_c" is the tasks' remote

communication time. Communication among processors is mostly small-grained data exchanging, which is very suitable to shared-memory communication mechanism. According to the Amdahl's Law, speedup can be represented as:

$$S = \frac{T_s}{T_p} = \frac{\sum_{k=1}^{n} t_s^k}{\sum_{k=1}^{n^1} t_s^k + t_c} . \quad (1)$$

On the ideal state, tasks are distributed evenly and there is no communication among tasks, so speedup is 2. But actually there are necessary data exchanging among tasks. According to the application characteristics on satellite, it can be estimated optimistically that when the processing program of task is parallelized fully (by static distribution or dynamic scheduling), the imbalanced execution time and cost of communication can be controlled ranging from 5% to 15% of T_s. From formula above, it can be concluded: (suppose the cost of communication is 10%):

$$S = \frac{T_s}{T_p} = \frac{\sum_{k=1}^{n} t_s^k}{\frac{1}{2}\sum_{k=1}^{n} t_s^k + 10\% \times \sum_{k=1}^{n} t_s^k} = 1.67 . \quad (2)$$

3.2 Fine-Grained Parallelism

The fine-grained parallelism is mainly applied to pay load processing, such as image transaction. The application speedup is decided mainly by the design of parallel algorithm. Here, a typical application of image processing is introduced as example to analyze the system's parallel capacity. In DFT algorithm, each output value of calculation is independent. All calculation tasks can be divided equally to each processor. There is little data-dependence between processors, so it is fit for this parallel design of close coupled.

The Amdahl's Law only evaluates parallel system performance from the aspect of task parallel degrees, its shortcoming lies in not containing various architecture characteristics of parallel processors, and uniform divisibility of task. But in fact, these actual factors are vital to the performance of the parallel processing. About above, advantages can be taken from the following timing model.

$$T_k = T_{kcomp} + T_{kcomm} + T_{ksync} + T_{kidle} \quad 1 \le k \le p . \quad (3)$$

Here, "T_{kcomp}" is the execution time that No. k unit used to completes it's distributed subtask; "T_{kcomm}" is the time of data communication with itself and other processing units; "T_{ksync}" is the necessary waiting time when there is data exchanging between multi-processors; "T_{kidle}" is the idleness waiting time of No. k processing unit that before the last subtask is finished by some processing unit. So, the time to finish all tasks — $T_{par} = \max\{T_k\}$, $1 \le k \le p$. The execution time using same algorithm on single processing unit is $T_{seri} = T_{1comp} + T_{2comp} + \ldots + T_{pcomp}$, thus the speedup of parallel processor is $S_p = T_{seri}/T_{par} \le p$.

3.3 The Consequence of Simulation Testing

Here gives two experiments, one is the computing of π to 12^{th} bit after dot, the other is a 64×64 matrix multiplying.

Table 1. Experiments of π computing and matrix multiplying

Experiment	π computing		Matrix multiplying	
# of CPUs	1	2	1	2
Calculation	To 12^{th} bit after dot	To 12^{th} bit after dot	64×64	64×64
T_{kcomp}	57265620 μ s	28881742 μ s	517884 μ s	260305 μ s
T_{kcomm}	0	1005104 μ s	0	5482 μ s
Sum of time	57265620 μ s	29886846 μ s	265787 μ s	265787 μ s
Speedup	1.916		1.948	

4 Conclusion

Based on analysis and simulation above, it can evaluated optimistically that adopting dual-CPU parallel architecture, the speedup can reach more than 1.6 compared with single-CPU system in embedded specific application.

There are two major problems in this system. Firstly, though each processor has its own local SRAM, with the increasing processors in the system, the load of shared-memory will increase rapidly. Besides, because bridge is employed instead of crossbar network, the scalability of system is constrained at the same time.

References

1. John L. Hennessy & David A. Patterson.: *Computer Architecture: A Quantitative Approach.* (2002)
2. Chenxi Zhang etc.: *Computer Architecture (Chinese).* (2000)
3. White paper: Practical System Design and Debug Considerations for Multiprocessing in the Embedded Environment. Broadcom Corporation. Printed in U.S.A. (2002)
4. Goodacre, J.: Understanding the Options for Embedded Multiprocessing. Information Quarterly. Vol. 2, No. 2 (2003) 33-39

Use Dynamic Combination of Two Meta-heuristics to Do Bi-partitioning*

Zhihui Xiong[1,2], Sikun Li[2], Jihua Chen[2] and Maojun Zhang[1]

[1] School of Information System and Management, National University of Defense Technology, 410073 Changsha, P. R. China
xzhnudt@vip.sina.com, maojun@mail.iscas.cn
[2] School of Computer Science, National University of Defense Technology, 410073 Changsha, P. R. China
lisikun@263.net.cn, jhchen@nudt.edu.cn

Abstract. In order to solve the hardware/software bi-partitioning problems in embedded system and System-on-a-Chip co-design, we put forward a novel bi-partitioning algorithm, which is based on the dynamic combination of Genetic Algorithm (GA) and Ant System Algorithm (ASA). The basic idea is: 1).Firstly, we use Genetic Algorithm to generate preliminary partitioning results, which are then converted into initial pheromone required by Ant System Algorithm, and finally we use Ant System Algorithm to search for the optimal partitioning scheme; 2).While the Genetic Algorithm is running, we determine the best combination time of GA and ASA dynamically, thus, the Genetic Algorithm avoids too early or too late termination. Experiments show that our algorithm excels GA and ASA in performance; moreover, we discover that the bigger partitioning problems are, the better our algorithm performs.

1 Introduction

Hardware/software partitioning is a fundamental and critical process in embedded system and SoC (System-on-a-Chip) co-design, which assigns system functions to certain hardware/software architecture optimally under some constraints. For a function set M = {m_1, m_2, ...,m_n}, a k-way partitioning tries to find out a cluster set P = {p_1, p_2, ...,p_k} that satisfies the following conditions:

$$\begin{cases} p_i \subseteq M, & 1 \le i \le k \\ \bigcup_{i=1}^{k} p_i = M & \\ p_i \bigcap p_j = \Phi, & 1 \le i, j \le k, i \ne j \\ r\text{esource and performance constraints} & \end{cases} \quad (1)$$

* Supported by National Nature of Science Foundation of China (90207019) and 863 Program (2002AA1Z1480).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 216-221, 2005.

If k=2, we call it bi-partitioning problem, in which case an embedded processor and some hardware accelerators are used.

Kastner made a comprehensive survey on system level partitioning in [1], some other typical researches include [2-8]. Based on Ant System Algorithm [9], Wang et al. presented a new approach for task level resource bi-partitioning problems in [10], which is a meta-heuristic method inspired by the study of the behaviors of ants. However, the Ant System Algorithm lacks a reasonable initial pheromone, which limits further improvement of performance.

Based on dynamic combination of GA [11] and ASA, we put forward an improved task level bi-partitioning algorithm. In this approach, we use Genetic Algorithm to generate preliminary partitioning results, then we convert these results into initial pheromone required by Ant System Algorithm, and finally we run Ant System Algorithm to search for the optimal partitioning scheme. Besides, we determine the best combination time of GA and ASA dynamically. Experimental results indicate that our algorithm provides a notable improvement.

2 Principle of Dynamic Combination of GA and ASA

After rigorous studies on the execution behaviors of Genetic Algorithms and Ant System Algorithms, we have found that these two types of meta-heuristic algorithms comply with the speed-time curves shown in Fig.1.

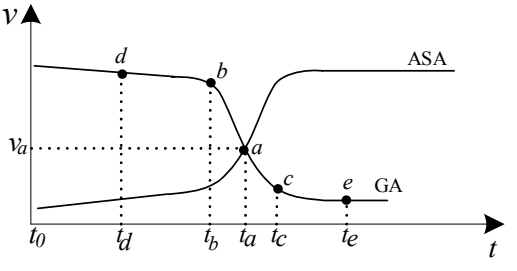


Fig. 1. Speed-time curve of GA and ASA

Genetic Algorithms run fast during time t_0 to t_a, after that, the searching efficiency decreases greatly; on the other hand, Ant System Algorithms run slowly during time t_0 to t_a because of the lack of initial pheromone, since the pheromone has to be accumulated to a certain strength (after time t_a), Ant System Algorithms converge to optimal results quickly. We call time t_a the optimal combination time.

To make use of the speed-time characters of GAs and ASAs, we consider dynamic combination of them, the principle is: 1).To get initial pheromone, we run Genetic Algorithms before optimal combination time t_a. 2).Then, after that, we run Ant System Algorithms with the initial pheromone generated. 3).In order to determine the optimal combination time t_a, we profile the colony fitness improvement ratio between iterations of Genetic Algorithms dynamically.

3 Bi-coloring Model of Bi-partitioning Problem

In order to represent hardware/software bi-partitioning problem, we use the bi-coloring model introduced in [10].

First of all, we use task graph to describe behaviors of embedded systems or SoCs. A task graph is a directed acyclic graph (DAG) G = (T, E), where T = (t_0, t_1, …, t_n) is a collection of task nodes, and E is a set of directed edges.

Then, we associate color c_1 (such as red color) with the task nodes assigned to software, color c_2 (such as green color) with hardware. So, what we need to do is to find the "optimal" coloring of task nodes using c_1 and c_2.

In order to associate the bi-coloring model with Ant System Algorithm, we further model the bi-partitioning problem as an agent (i.e. ant) based stochastic decision making process as detailed in [10]. An agent tries to construct a coloring on the task graph based on the distributed and local heuristics.

For every task node t_i, two steps are carried out by the agent. 1).The agent makes a decision on how the task node t_i should be colored. This decision is made by considering the choices made on t_i by its immediate predecessors. Once this decision is made, the agent forces every inbound edge of t_i to be colored in the same color as t_i. 2).The agent guesses the color of each task node that immediately follows t_i, i.e. for every edge e_{ij}, it guesses the color of t_j. The probability of assigning color c_k to t_j is based on a global heuristic $\tau_{ij}^{C}(k)$ and the local heuristic, while the local heuristic is based on the local information from task nodes t_i and t_j .

4 Our Hardware/Software Bi-partitioning Algorithm

Similar to [10], we consider task level bi-partitioning and we use bi-coloring to model the problem. There are three parts in our algorithm, i.e. the Genetic Algorithm part, the joint part of the GA and ASA, the Ant System Algorithm part.

4.1 Rules on Genetic Algorithm

The rules on GA part of our algorithm include:

- Genetic Encoding: Use binary encoding [12]; 0 stands for color c_1 (software), and 1 stands for color c_2 (hardware).
- Object Function and Fitness Function: We seek for the shortest running time under area constraints, so we define object function as $s_{best} = \arg\min time_s$, and we define fitness function as $fitness = 1/time_s$, Where $time_s$ denotes the run time of coloring (partitioning) scheme s .
- Generation of Initial Colony: Use random method to generate initial colony.
- Selection Operator: We adopt the widely used Roulette Wheel Selection policy [12].
- Crossover Operator: Use the Uniform Crossover policy, which exchanges each bit of the two parent's code string with certain probability.

- Mutation Operator: Bit reverses a selected gene (individual) by mutation probability p_m.
- Controlling Parameters: Our algorithm performs operator non-overlap genetic operations. We set colony size $N = 50$, crossover probability $p_c = 0.6$ and mutation probability $p_m = 0.2$.
- Terminate Conditions: In fact, they are consistent with the optimal combination time. We set minimum iterations $Gene_{\min} = 15$, maximum iterations $Gene_{\max} = 50$, iteration minimum improvement ratio $Gene_{\min-impro-ratio} = 3\%$ and $Gene_{die} = 3$.

4.2 Rules on Ant System Algorithm

In order to make comparison, we set Ant System Algorithm rules the same as those of [10] as far as possible. The main differences are:

- Pheromone Setting and Refreshing: We adopt MMAS (Max-Min Ant System) introduced by Stutzle [13], the refreshing equation of pheromone is:

$$\tau_{ij}(k) = \begin{cases} (1-\rho) * \tau_{ij}(k) + \Delta\tau_{ij}(k)_{best} \\ \tau_{ij}(k)_{\max}, \text{if } \tau_{ij}(k) > \tau_{ij}(k)_{\max} \\ \tau_{ij}(k)_{\min}, \text{if } \tau_{ij}(k) < \tau_{ij}(k)_{\min} \end{cases} \quad (2)$$

  Where ρ is the evaporation ratio of pheromone, $\tau_{ij}(k)_{\max}$ ($\tau_{ij}(k)_{\min}$) is maximum (minimum) strength of c_k pheromone on edge e_{ij} (we set them ∞ (60)), and $\Delta\tau_{ij}(k)_{best}$ is the increment of c_k pheromone on edge e_{ij} done by the best ant in current Ant System Algorithm iteration.
- Terminate Conditions: The Ant System Algorithm terminates if one of the following two conditions is satisfied: 1).Ant System Algorithm iterates $Ant_{\max}$ times. 2).Child iteration fitness improvement ratio is continually lower than $Ant_{\min-impro-ratio}$ for Ant_{die} iteration(s). We set $Ant_{\max} = 100$, $Ant_{die} = 3$, and $Ant_{\min-impro-ratio} = 0.5\%$.

4.3 Joint of the Two Algorithms

The joint of GA and ASA is important in our approach; we mainly consider the following problems:

- Optimal Combination Time: This is the same as the Terminate conditions described in "Rules for Genetic Algorithm".
- Initial Pheromone Setting: Wang [10] set initial pheromone with a fixed value $\tau_0 = 100$. We improve this by using Genetic Algorithm to get more reliable values. In our algorithm, initial pheromone on edge e_{ij} is determined by $\tau_{ij}^S(k) = \tau_{ij}^C(k) + \tau_{ij}^G(k)$, where $\tau_{ij}^C(k)$ is a constant which indicates the c_k pheromone value on edge e_{ij}, this value corresponds to $\tau_{\min}$ in MMAS [13]. $\tau_{ij}^G(k)$ is the c_k

pheromone on edge e_{ij}, which is converted from the searching results of Genetic Algorithm. We set $\tau_{ij}^{C}(k) = \tau_{ij}(k)_{\min} = 60$, $0 \le i, j \le n$, $k=1, 2$.

- Conversion from GA Results to Initial Pheromone: In the last generation of GA, we select the top 10 percent best individuals to construct a genetic optimization results set (denoted with $S_{10\%better}^{gene}$. Besides, we set initial value of $\tau_{ij}^{G}(k)$ as 0, $0 \le i, j \le n$, $k=1,2$. For each result s in $S_{10\%better}^{gene}$, if the edge e_{ij} is colored c_k, then $\tau_{ij}^{G}(k)$ increases 20 by itself.

5 Experiments

We use a similar experimental model to that of [10], and make comparison with the partitioning method that is based on GA [7] and the method that is based on ASA. Table 1 shows the data we have obtained. In order to make the comparison valid, we use the same controlling parameters for these three types of partitioning algorithms. In order to compare the algorithms' speed performance, we have restricted that the three algorithms process execution until the results have the same distance to the theoretical optimal results.

To construct the experimental samples, we generate seven groups of DAGs (denoted by DAG_{20}, DAG_{30}, $\cdots$, DAG_{80} respectively, the subscripts stand for total number of nodes in each graph). There are 30 DAGs in each group, and the average branching factors of the seven DAG groups are 5,5,7,7,9,9,11. We use the average performance values of the thirty DAG samples as the final performance results.

In our experimentation, we use the following PC configurations: a).AMD Duron 700MHz processor, 128MB memory. b).Redhat Linux 7.3 operating system. c).KDevelop 2.1 programming tools.

Table 1. Comparison of GA, ASA and our algorithm

Total DAG Nodes	GA [7]		ASA [10]		Our Algorithm	
	Time (ms)	Iterations	Time (ms)	Iterations	Time (ms)	Iterations
20	258	72.3	332	43.2	207	23.1 + 14.6
30	579	65.1	659	36.8	513	31.2 + 12.4
40	1296	83.6	1463	47.3	742	19.4 + 21.6
50	2694	89.2	2235	44.7	1662	27.7 + 18.3
60	4833	92.7	3280	68.7	2038	26.8 + 21.8
70	8436	68.0	5762	51.3	2679	21.4 + 19.4
80	15623	97.5	9368	55.1	3651	27.2 + 16.3

We can draw the conclusion from the table that, when the size of task graph (total number of nodes) is relatively small (20~30 nodes), our algorithm performs almost the same as GA and ASA. However, when the total number of nodes in the task graphs exceeds 40, our algorithm performs better (faster). This is because when the number of nodes increases, the searching space of partitioning problems grows up exponentially, but our algorithm is not so sensitive to the increase of the search space.

6 Conclusions

A novel partitioning algorithm based on dynamic combination of Genetic Algorithm and Ant System Algorithm is introduced in this paper. This algorithm takes the advantages of the two algorithms and overcomes their disadvantages. Experiments show our algorithm excels GA and ASA in performance.

References

1. Kastner, R.: Synthesis techniques and optimizations for reconfigurable systems. Los Angeles: University of California, 2002
2. Gupta, R., Micheli, G.D.: System-level synthesis using re-programmable components. Proc. of the Euro. Conf. on Design Automation, (1992) 2-7
3. Ernst, R., Henkel, J., Benner, T.: Hardware-software co-synthesis for microcontrollers. IEEE Design and Test of Computers 10 (1993) 64-75
4. Vahid, F., Jie, G., Gajski, D.D.: A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. Proc. of the Euro. Conf. on Design Automation, (1994) 214-219
5. Niemann, R., Marwedel, P.: Hardware/software partitioning using integer programming. Proc. of the Euro. Design and Test Conf., (1996)
6. Kalavade, A., Lee, E.A.: The extended partitioning problem: hardware/software mapping, scheduling, and implementation-bin selection. Design Automation for Embedded Systems 2 (1997) 125-163
7. Saha, D., Mitra, R.S., Basu, A.: Hardware Software Partitioning using Genetic Algorithm. Proc. of the 10th Int'l Conf. on VLSI Design (1997) 155-160
8. Wiangtong, T., Cheung, P., Luk, W.: Comparing three heuristic search methods for functional partitioning in hardware-software codesign. Journal of Design Automation for Embedded Systems 6 (2002) 425-449
9. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. on Systems, Man and Cybernetics, Part-B, 26 (1996) 29-41
10. Wang, G., Gong, W.R., Kastner, R.: A new approach for task level computational resource bi-partitioning. Proc. of IASTED Int'l Conf. on Parallel and Distributed Computing and Systems (2003)
11. Holland, J.H.: Adaptation in natural and artificial systems. Michigan University Press (1975)
12. Pan, Z.J., Kang, L.S., Chen, Y.P: Evolutionary Computation. Beijing: Tsinghua University Press (1998)
13. Stutzle, T., Hoos, H.: MAX-MIN ant system. Future Generation Computer System 16 (2000) 889-914

A New Approach for Predictable Hard Real-Time Transaction Processing in Embedded Database*

Tianzhou Chen, Yi Lian, Jiangwei Huang

College of Computer Science, Zhejiang University,
Hangzhou, Zhejiang, 310027, P. R. China
{tzchen, yl, hjw}@zju.edu.cn

Abstract. Real-time transaction processing becomes concerns as embedded time comes rapidly. However, the transaction process in real-time embedded system still has some problems in resource utilization at present. In this paper, a classification is designed to differentiate the hard periodic real-time transactions and hard sporadic real-time transactions. The new processing algorithm for hard sporadic real-time transactions is given. In time consuming and space consuming, this design and implementation indeed do a great deal of good.

1 Introduction

Real-time database has improved the storage and manipulation of data in embedded system these years. Most embedded database systems are also real-time. We refer to these systems as real-time embedded database. Transactions in the system must be scheduled in such a way that they can be completed before their corresponding deadlines expires as well as satisfy database consistency constraint.

Real-time transactions can be grouped into three categories: hard deadline, firm deadline, and soft deadline. [1] For a hard deadline transaction, missing a deadline is equivalent to a catastrophe. For firm or soft deadline transaction, however, missing deadline leads to a performance penalty but does not entail catastrophic result.

A lot of previous researches focus on transaction processing with pure deadline monotonic scheduling. This method indeed enhances the utilization of the system resource and guarantee the urgent transaction achieve. But this method cannot guarantee all hard real-time meet its deadline. At the same time, some researches on real-time transaction process focus on static pre-scheduled approach, which has some problems. One of these problem is inefficiency: the system does not maximize the use of the processor. A lot of processing time must be reserved for handling an event, even though in reality that time is hardly ever used for hard real-time processing.

Major problem of previous researches is that they don't consider how to cooperate real-time and embedded system. As known, the flash acts as storage in most embedded system. The maximum time of the I/O is up to 100000. Furthermore, the

* This work is supported by the national 863 high technology project and HP Embedded Laboratory of Zhejiang University

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 222-228, 2005.

embedded systems often work in remote area, which is not easy to maintain. Hence, besides real-time requirement, another goal must be to decrease the access to flash devices.

Observe that a real-time model [1] (called Kim & Son's model following) do well in transaction classification and responding process scheme. However, it is designed for real-time database, not embedded conditions considered. Hence, it is not very suitable to embedded database, because of too much access to flash device. To solve this problem, we modify two of transaction types of it and develop two new type transactions and responding process scheme.

2 Kim & Son Model Performance Analysis and Modification

Kim & Son model gives the classification of the hard real-time transaction in conventional database based on the nature of transaction in real-time database systems. This classification is summarized in Figure 1, except the new types.

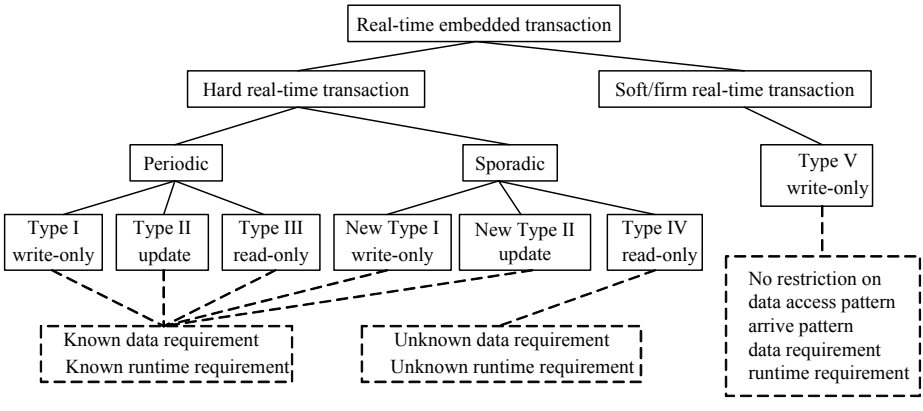


Fig. 1. New classification of real-time embedded transactions

Suppose the classification applies to real-time embedded database. As known, the flash acts as storage in most embedded systems, especially mobile devices. As known, the space in flash storage device is often limited. Suppose 10 type I transactions are running, each transaction writes 10 bytes every second. Nearly 10M data will accumulate in flash one night. The flash storage device may expire in a week. Another problem is that block is the minimum flash erased unit, and flash won't suffer from frequent erasing. The times of flash erased are up to 100000. Furthermore, type II transaction comes periodic at the same way. This situation is aggravated.

The question comes up. Is it necessary to excuse the type I & type II transactions periodic, writing every value from an external object to embedded database? Let us consider a mobile medical information system as an example: Type I transactions are transactions which write the dynamic physical status of a critical patient in motion in the open from the sensor devices, such as heart beat sensor. Obviously type I transaction (e.g. the heart beat of the patient) will go steadily rather than change up

and down frequently at most of the time. The derived data are more or less the same. The necessary time the record need flushing into flash storage is at the checkpoint or at the time when some abnormal data come. And the transaction only needs to write the criterion function (e.g. the average value) to the flash device. When it comes to type II transaction, the type II transactions can work only in key point such as when the critical event happens to the patient. Hence, for the special status of embedded real-time database, therefore, we do some modification on Type I and Type II, from which we develop two new types, summarized by Figure 1.

New Type I & II, transaction of these types are picked up from the corresponding Type I and Type II transactions, simple to previous transactions, which access the flash with hard deadlines, and their runtime and data requirements are known in advance. But they are not periodic. They only write the key data at the checkpoint time or when some abnormal data come, rather than always.

It is believed that the classification is more efficient in processing transactions, and improves overall system performance. Hence two new types of transactions appear. Fortunately they are similar; both of them are sporadic, with known data requirement and runtime requirement. Following is the corresponding process scheme.

3 Predictable Sporadic Transaction Processing Implementation

The transactions we will process are all sporadic, with known data requirement and runtime requirement. Thus in an efficient real-time embedded database, two points below must be concerned:

For one thing, since the limited resource in embedded system, the deadline monotonic scheduling may be the first choice. But the major problem of this scheduling is that it can't guarantee all sporadic transactions to meet its deadline. For another, it's important that systems operate correctly because the costs of even trivial failures are high.

Before we can go through the computation, we need to make some assumptions about the transaction. First, there is a fixed set of transactions with known data requirement and known process runtime; Secondly, each time a transaction becomes ready to run, it will run for only a bounded amount of processor time; Thirdly, before a transaction completes, it won't be restarted. Finally, the transaction with shorter deadline has the higher priority. So the implementation in this paper uses a priority-based algorithm to compute the worst-case response time of every new type transaction, to filter some transaction out, and to guarantee remain transactions meet their deadline with deadline monotonic scheduling.

Our first step is to find the worst-case response time from the time the transaction *i* gets ready to the time the transaction completes. So we want to find an equation that will calculate the worst-case response time of the ith transaction denoted by R_i, made up of two parts: the worst-case execution time a transaction takes on its own, denoted by C_i, and the time it takes to wait for higher priority transaction to execute, denoted by W_i. The following equation represents this relationship:

$$R_i = C_i + W_i \quad (1)$$

Here the deadline of a transaction i is denoted by D_i, and a transaction must always meet its deadline, namely $R_i \leqslant D_i$. Thus, the problem now becomes finding the worst-case waiting time W_i. The term P_j is the so-called "the minimum interval" of two same kinds of transactions. Although *new type transactions* are sporadic, the interval time between two transactions of the same kind is integral times of the original periods of primary type. *(i.e., $P_j \geqslant T$ T= period of primary type).* The total time that a given transaction i may be waiting while transaction j is ready is given by:

$$\left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j \tag{2}$$

The first factor above is the time taken by a transaction j with higher priority when it pre-empts and executes. Thus the value that adds this up for all the higher priority transactions j, $j \in h_i$ (h_i is the set of all transactions of higher priority than transaction i) is the total time transaction i waiting in the system and get the following:

$$W_i = \sum_{\forall j \in h_i} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j \tag{3}$$

We can integrate equation (1) and (3) and present the response time equation

$$R_i = C_i + \sum_{\forall j \in h_i} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j \tag{4}$$

Because R_i appears on both the left- and right-hand sides of the equation, it is difficult to work it out directly. But it is easy to construct a process. Suppose a transaction i responses in $R_i(n)$. At fact, in $R_i(n)$, some higher priority transactions may come, in result, the response time prolongs to $R_i(n+1)$. Hence a recurrence relation is formed to solve this problem:

$$R_i\,(n+1) = C_i + \sum_{\forall j \in h_i} \left\lceil \frac{R_i\,(n)}{P_j} \right\rceil C_j \tag{5}$$

Step by step, if a value of $R_i(n)$ satisfies the equation (4), it must be the smallest, which is the first time when transaction i has enough response time to complete, i.e., the worst-case response time. And initialize the first value of R_i to zero and the sequence will converge to the smallest value of R_i that satisfies Equation (4).

Let's have an example. Table 1 describes a set of transactions in *New Type*. The transaction in table is in deadline monotonic priority order, with transaction 1 being the highest priority and transaction 4 the lowest priority. Calculate the worst-case response time of transaction 3. We start with an initial R estimate of 0. Table 2 shows the steps in the calculation. The equation converges at 38ms. So the worst-case response time of transaction 3 is 38ms. This is less than the deadline of 50ms and proves that transaction 3 will always meet its deadline in all situations.

Table 1. Deadline Monotonic Analysis Example

TRANSACTION	P	C	D
1	250ms	5ms	10ms
2	10ms	2ms	10ms
3	330ms	25ms	50ms
4	1000ms	29ms	55ms

Table 2. Calculation of worst-case transaction 3 response time

STEP	R_n	W	R_{n+1}
1	0	0	25
2	25	5+3*2	36
3	36	5+4*2	38
4	38	5+4*2	38

So far, it is known that which some of *new type transactions* (transaction 4 in above example) can't meet their deadline with deadline monotonic. Since these transactions are picked up from the type I and type II transactions, let them back to their primary type, periodic, using the primary process scheme. [1] Just as said that simple is beauty.

4 Performance Analysis and Discussion

In the above chapter, two new type transactions are added to new model. Are they minority in a practical system? According to the investigation, most transactions in type I and type II can be translated into the new type transaction. Quote the above mobile medical information example, the data derived from the heart beat rate, temperature sensor, and most apparatus sensor can be translated to new type transaction. Furthermore, these transactions quoted are great deal data producers.

And it is mentioned that the *new type transactions*, which are not suitable to the new process scheme, must be back to their primary types. Are most new type transactions suitable to new type process scheme?

It is a misconception that "hard real-time" means "no way to consult". In the design phase, all functions for the system is known and the design of the deadline is not only referred to the attributes of the data, but also referred to the worst-case response time, therefore, these deadlines must be designed appropriately in advance. Each time a new scheme draft is developed, we can use the above the algorithm to compute the worst-case response time of every transaction. Adjust most of all work.

This translation has at lease two advantages following.

First of all, decrease the production of the data and access to flash storage distinctly, since most amounts of data from type I and type II transactions are translated to new type transaction:

Suppose in a normal embedded real-time system, the primary type I & type II transactions produce more than 60 percents of the total amount of the data, denoted by $D_{I\&II}$. The percentage of *type I & type II transactions* which are able to be translated to *new type transactions* is denoted by *NP*. Suppose the values of the target data *x* are in normal distribution. And the mathematical expectation μ is zero; the mean square σ is 1; Namely,

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}} e^{\frac{x^2}{2}} \quad \textbf{(6)}$$

If the new value x_{new} is *Diff* more or less than μ , it is regard as an abnormal value. The table 3 and table 4 are the percentage of the total data with 60% and 80% primary *type I & type II transactions* won't write to the embedded database using new transaction processing in different *NP* and different *Diff*:

Table 3. the percentage of total data with 60% primary *type I & type II transactions* won't write to the embedded database using new transaction processing

NP \ Diff	0.5σ	σ	2σ
50%	11.5%	20.5%	28.7%
60%	13.8%	24.6%	34.4%
70%	16.1%	28.7%	40.1%
80%	18.4%	32.8%	45.8%
90%	20.7%	36.9%	51.5%

Table 4. the percentage of total data with 80% primary *type I & type II transactions* won't write to the embedded database using new transaction processing

NP \ Diff	0.5σ	σ	2σ
50%	15.3%	27.3%	38.3%
60%	18.4%	32.8%	45.9%
70%	21.5%	38.3%	53.5%
80%	24.5%	43.7%	61.1%
90%	27.6%	49.2%	68.7%

Secondly, periodic transactions call for static pre-schedule, the system does not maximize the use of the processor. For example, A new kind of transaction is added,

although there might be quite a lot of spare time across the schedule, there's not enough spare time in any one place; a transaction has to fit within the spare time in a single time slot. When this transaction executes for longer than the spare time in any slot, there must be restructuring of the source code.

5 Conclusions

In a real-time embedded database application, above all, the scheduling algorithm must guarantee that all the hard deadline transaction will complete by their deadlines and then try the best with the remaining soft or firm deadline transactions. Secondly, it must guarantee the utilization of the system resource and minimize the access to flash storage device. This goal will be benefited from our new embedded real-time database model.

In this paper, we translate a real-time data and transaction model to real-time embedded database model and provide an improved framework to realize the predictable real-time embedded transaction processing. However, we cannot guarantee all the soft or firm deadline transaction, unknown data requirement and unknown runtime requirement meet their deadline. We observe that no transaction scheduling algorithms proposed so far address this problem completely even though many research papers in the real-time embedded database field have pointed it out. Hence, the goal of our research is to investigate a proper model for real-time embedded data and transactions and an adequate software and hardware architecture for real-time embedded database system.

Future work includes implementing a real-time embedded database with the proposed architecture on our experimental board, and improving the functionality of storing and manipulating data objects in our real-time embedded system.

Reference

1. Young-KuK Kim, Sang H.Son An approach towards predictable real-time transaction processing (1993)
2. Pao-Ann Hsiung, Cheng-Yi Lin, Synthesis of real-time embedded software with local and global deadlines, 2003
3. Aleksandra Tesanovic, Dag Nystrom, Jorgen Hansson, Christer Norstrom Embedded Databases for Embedded Real-Time Systems:A Component-Based Approach 2002
4. Anindya Datta Providing real-time response state recency and temporal consistency in database fro rapidly changing environments.
5. F.Eirayes, J. Rolia, and R. Friedrich The performance impact of workload characterization for distributed applications using ARM.2000
6. Mario Baldi, Yoram Ofek, A comparison of ring and tree embedding for real-time group multicast. 2003
7. Thomas A. Henzinger, Christoph M. Kirsch, The embedded machine: predictable, portable real-time code. 2002

A QoS-aware Component-Based Middleware for Pervasive Computing*

Yuan Liao and Mingshu Li

Internet Software Technology Lab, Institute of Software,
Chinese Academy of Sciences, Beijing, P.R.China. 100080
{liaoy, mingshuli}@intec.iscas.ac.cn

Abstract. In this paper, a QoS-aware component-based middleware for pervasive computing is given. The design rationale is to cope with many constraints of pervasive computing application including resource awareness, diversity, QoS-sensitive. To introduce how we achieve those system goals, we describe the details of this middleware including system architecture, service model, QoS model, algorithms of service setup, implementation, and experimentation.

1 Introduction

It is very meaningful to design and implement a middleware support for applications in pervasive computing environments. In pervasive computing environments, middleware developer would meet many design challenges including diversity and heterogeneity, resource-awareness, QoS-sensitive [2, 9]. To tackle those requirements, in this article, we design a component-based and QoS-aware middleware-QuCOM (Quality COMponent) for pervasive computing application. In QuCOM, a service is a composition of components that have QoS specification defined in our proposed QoS model. When a service is deployed, QoS requirements and the corresponding resource allocations of service component should be satisfied by invoking QoS-aware resource management mechanisms of system. Therefore, our solution spans from QoS specification in the development phase of an application service, to QoS enforcement and resource management at runtime.

QoS management has been widely discussed in the area of middleware systems[1], and QoS-aware middleware systems have emerged to assist a new spectrum of applications that require QoS in pervasive computing environments, for example, $2K^Q$[6], Agilos[4], TAO[7], QuO[8], QoSME[3], QoS management framework of Gaia OS[2, 11], and Q-RAM[5]. However, component-based middleware in QoS-aware middleware is very limited since QoS-sensitive application has little attention in component-based software. Most work is based on object-oriented technologies, for example, TAO, QuO, $2K^Q$, etc. The lack of suitable QoS model for component-based system is

* Research supported by the national high-tech research and development plan of China under Grant No: 2002AA1Z2302 and 2004AA1Z2050.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 229-235, 2005.

one of major reasons. The remainder of this paper will show the details of QuCOM including system architecture, service model, QoS model, algorithms of service setup, implementation, and experimentation.

2 QoS-aware Resource Management Architecture

2.1 System Architecture

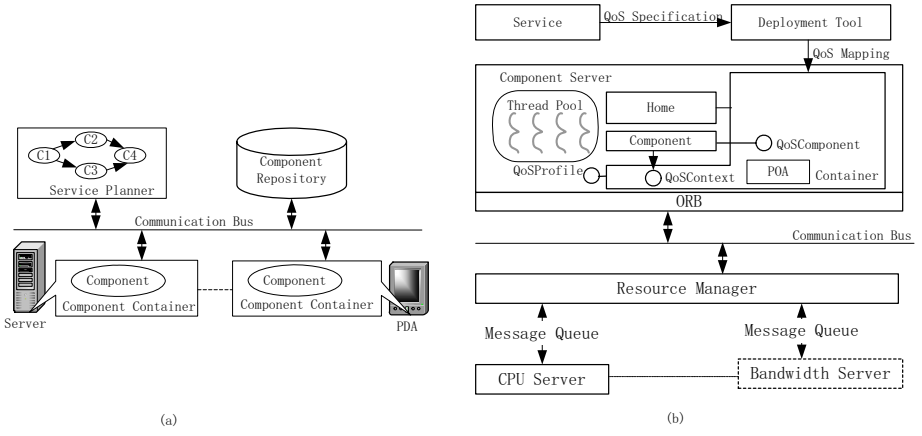


Fig. 1. Horizontal and Vertical view of QuCOM's architecture

Fig.1 illustrates the horizontal and vertical view of QuCOM's architecture from QoS perspective. First of all, service is made up of components, which is described by component specification including QoS and resource parameters defined by component designer at development phase. At deployment time, those application-level QoS parameters in component specification would be mapped into system-level QoS (e.g. CPU deadline,period etc) enforced by QuCOM. QuCOM is implemented based on CCM (Corba Component Model) middleware, which has four core components in QuCOM system, namely, component container, service planner and component repository.

The component container is an execution environment that provides the resources required to execute components. And to guarantee the QoS of component, it is responsible for resource reservation, QoS enforcement, and QoS adaptation. Component containers can be implemented as processes running in heterogeneous environment, such as Linux server, vxWorks embedded system, and WinCE PDA, which satisfies the diverse and heterogeneous environments constraints of pervasive computing application.

The component repository allows providers to register their component descriptions and store components. Component descriptions include not only functional interface description but also elements that describe the QoS and resource requirements of component.

The service planner is used to setup service. When an instance of service is initiated, the service planner contacts the component repository to set up the service by selecting appropriate mode of service components (to be described in 2.3). In execution phase of service, service planner is used to adapt the instance of the service to runtime resource fluctuations by revising the mode of service components.

The resource servers reserve and allocate resource for QoS guarantee. During the setup and execution phase of service, to guarantee the QoS of service component, the component container contacts the resource server to allocate resources required. Those resources are reserved and enforced by relevant resource server, such as CPU server, communication server and memory server etc. For example, DSRT [12] can be used to set up and enforcement CPU reservation and network bandwidth can be reserved in RSVP-enable framework. From an implementation perspective, those resource server are processes in user layer of OS and can be implemented in many OSes easily.

2.2 Service Model

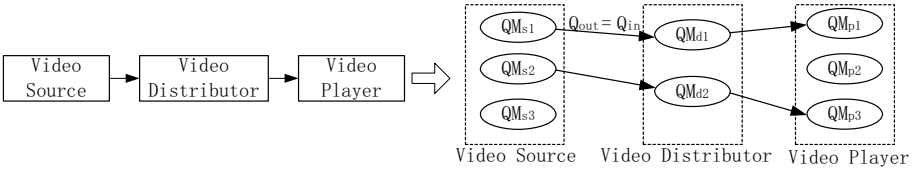


Fig. 2. Video Streaming Service and its QoS graph

A service is modeled as a collection of service components combined according to control-flow and data-flow dependencies. In this paper, to simplify the discussion, service is described by DAG (Directed Acyclic Graph) diagram, which is made up of components and transitions. Transitions of a DAG are labeled with events or interfaces. Nodes of DAG are atomic components. For example, a video streaming service is composed of video source component, which retrieves and transmits videos stored in the server; video distributor component, which transcodes video format between video source and video player; and video player component, as shown in Fig .2.

To capture the component QoS and resource requirement to achieve the QoS, we define a QoS model of service component and service. QoS specification and mode are described as follows:

Definition 1: QoS specification of a service component *c* is made up of QoS modes and QoS transitions, i.e. $QoSSpec_c$=(*ModeList, TransitionList*).

- *ModeList* is a list of QoS modes of *c,* i.e. $ModeList=\{<Mode_1, Weight_1>, \ldots <Mode_i, Weight_i>, \ldots, <Mode_n, Weight_n>\}$. $Mode_i$ is a QoS mode, and component designers set $Weight_i$, which represents the weight of $Mode_i$.
- *TransitionList* is a list of transitions between QoS modes of *c. TransitionList*=$\{Transition_1, \ldots, Transition_i, \ldots, Transition_n\}$. $Transition_1$ is a transition to be called when QoS mode of component *c* is changed.

Definition 2: QoS mode is associated with an input QoS vector QoS_{in}, an output QoS_{out}, and resource requirement vector *Res*, i.e. $Mode_i = (QoS_{in}, Res, QoS_{out})$.

- Both QoS_{in} and QoS_{out} are QoS vectors of multiple application-level QoS parameters. Therefore, they are enumerable and have the form $[q_1, q_2,\ldots, q_m]$.
- *Res* is a vector of resource requirement, $[r_1, r_2,\ldots, r_k]$, in which r_j $(1\le j\le k)$ is the amount of the *j*th resource required by service component *c*.

As we known, service is represented by a dependency DAG. From QoS perspective, nodes in the DAG represent the participating service components, while edges in the DG represent the input/output and QoS dependencies between the service components. An edge from service component C_1 to C_2 indicates that the output of C_1 is the input of C_2; and the QoS_{in} of C_2 is satisfied by the QoS_{out} of C_1, which means that each QoS parameter, the corresponding value in QoS_{in} of C_2 is equivalent to QoS_{out} of C_1. If all of service component's QoS is satisfied by their predecessor in DAG and their resource requirement can be reserved, the service can be set up (to be described in 2.3); then QoS_{in} of the service is equivalent to QoS_{in} of the initial component node, QoS_{out} of the service is QoS_{out} of the final component node in DAG of service.

2.3 Service Setup

As defined in QoS model, each service component may have multiple QoS modes associated with a certain resource requirement vector. When an instance of service is initiated (i.e. service setup), the service planner should select a QoS mode for each service component so that the mode has the highest possible weight value; and its QoS_{in} is satisfied. This is under the constraint that its resource requirement should be satisfied by current resource availability. There are two steps during service setup: first the service planner constructs a QoS graph based on QoS model of this service, then selects appropriate QoS mode for each service component based on QoS graph and available resource of system.

For each participating service component *c*, all of its QoS modes are represented as nodes of QoS graph, as shown in Fig.2. In QoS graph, QoS modes of a service component are partially ordered by their weight. Edges from one QoS mode QM_i to other QM_j exist, if and only if QoS_{out} of QM_i is equivalent to QoS_{in} of QM_j.

After constructing the QoS graph, the service planner will select appropriate QoS mode for each service component. The selection is based on two constraints: the resource requirements of selected service component are satisfied by available resource and the QoS mode of highest possible weight should be selected. Firstly, we find all of paths from the initial node to final node of QoS graph and calculate resource requirement of each path, which is sum of path component's resource requirement:

$$Res_{path}=[\sum_{i=1}^{N} r_1,\ \sum_{i=1}^{N} r_2,\ldots,\ \sum_{i=1}^{N} r_k\ ]. \quad (1)$$

where k is number of resource category and N is component number. Those are candidate paths. If and only if Res_{path} can be satisfied, i.e.:

$$\sum_{i=1}^{N} r_j \le r_{avail}\ (1\le j\le k). \quad (2)$$

path is feasible resource reservation path. There are two cases:

1) There possibly exists more than one feasible path. To make a choice, those candidate paths are sorted with weight of their final node. The service planner selects the path that has maximum weight of final component node in those candidates, and then makes resource reservation for all components of the path.

2) The service planner can't find out one feasible path for the new incoming service s_{new}. In this case, we use QoS negotiation mechanism as a way to grant the service admission. This mechanism needs services to record their candidate paths and resource requirements. Let a set of pairs $ps=\{<Path_1, Res_{path1}, Weight_1>, < Path_2, Res_{path2}, Weight_2>,\ldots, < Path_n, Res_{pathn}, Weight_n>\}$ represents the candidate path set of a service s, where $Path_i$ is a candidate path , Res_{pathi} is its resource requirements as defined by (1) and $Weight_i$ is its final component node weight; If there are m services that have been set up in the pervasive computing environment, s_{new} needs a negotiation with the m services about resource when s_{new} can't be set up. The method is to select a candidate pair p from the path set ps for each service including m services and s_{new}, such that $Res_{p.path}$ can be satisfied and make the sum of selected pair's weight be maximum. Therefore, the selection algorithm can be based on Integer Programming (IP) [10]. The objective function is:

$$\text{Max} \left(\sum_{i=1}^{m+1} Weight_i \right). \tag{3}$$

The constraint of the IP problem is that all selected pair's paths are feasible resource reservation paths.

3 Experimentation

In order to evaluate QuCOM, we have developed a video streaming service including three service components: video source, video distributor, and video player, as shown in Fig.2, deployed on the same configuration PCs with different OS. The configuration of PC is: Pentium IV 1.4GHz with 265M RAM. Video source and video distributor are running in different Linux server, while video player is in Windows 2000. They are connected to a LAN through 100Mbits/sec Ethernet card.

Table.1. QoS specification of service components

Component	Weight	Mode	QoS_{in}	Resource	QoS_{out}
Video distributor	150	$Mode_0$	*fps* =20	Res_{cpu}=20%	*fps* =20
	100	$Mode_1$	*fps* =10	Res_{cpu}=10%	*fps* =10

In the experiment, we will change the available resource of CPU of host which video distributor is running on, by adding load-simulating processes at t=45, 90, and 135 seconds, each attempting to use 25% of CPU resource. And total amount of CPU resource that can be allocated for QoS guarantee is 90%. Due to space constraint, we

will only provide QoS specification of video distributor as table.1, where QoS_{in} and QoS_{out} are frame rates of incoming and outgoing video streaming.

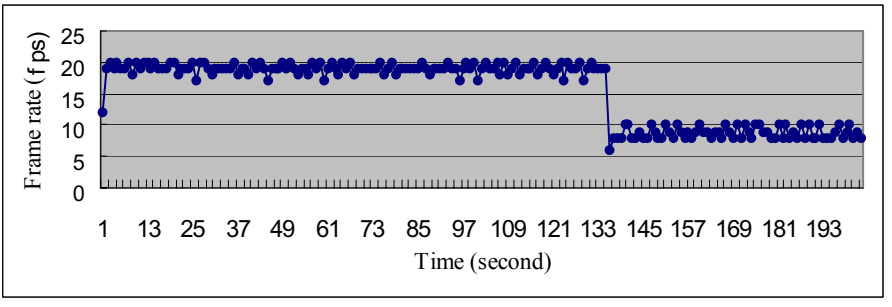


Fig. 3. Actual QoS_{out} of video distributor component

Fig.3 shows the actual QoS_{out} of video distributor component instance as available resource of CPU is decreasing. In this graph, the x-axis represents the passage of time in seconds from the component is deployed in a component server, and y-axis represents frame rate of outgoing video streaming, i.e. QoS_{out}. From this graph, we can see that frame rate is stable from t=0 to t=135 seconds, and from t=135 seconds to t=200 seconds. At t=135 seconds, frame rate changes from 20 *fps* to 10 *fps* since the 3rd load-simulating processes attempts to use resource, which triggers QoS negotiation. The outcome of this experiment demonstrates that QoS parameter of accepted service component can be guaranteed since its resource can be satisfied and QoS negotiation will make QoS mode of component change.

4 Summary and Future Work

In this paper, we present QuCOM, which is a QoS-aware component-based middleware support for pervasive computing. In QuCOM, those essential works of QoS-aware middleware including QoS specification, setup, enforcement, and resource management are all covered. Based on the results from our development, we believe that our solution can cope with many constraints of pervasive computing application, such as resource awareness, heterogeneity and QoS-sensitive. However, at present we don't deal with resource fluctuations that some application is very sensitive to. In the future, our work will be focused on QoS adaptation to fluctuating resource based on feedback model.

References

1. C.Aurrecoechea, A.T.Campbell, and L.Hauw.: A Survey of QoS Architectures. Multimedia Systems, Vol.6, no.3, pp. 138-151, 1998.

2. K. Nahrstedt, D.Xu, D. Wichadakul, and B.Li.: QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. IEEE Comm, Magazine, Vol. 39, no.11, pp. 2-10, 2001
3. Patricia Gomes Soares Florissi.: QoSME: QoS Management Environment. Ph.D. Thesis, Columbia University, 1996.
4. B. Li and K. Nahrstedt.: A Control-based Middleware Framework for Quality of Service Adaptations. IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms, vol. 17, no. 9, pp. 1632–1650, Sept. 1999.
5. R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek.: A Resource Allocation Model for QoS Management. In Proceedings of the IEEE Real-Time Systems Symposium, pp. 298–307, Dec. 1997.
6. K. Nahrstedt, D. Wichadakul, and D. Xu.: Distributed QoS Compilation and Runtime Instantiation. In Proceedings of the Eighth IEEE/IFIP International Workshop on Quality of Service, pp. 198–207, June 2000.
7. D. Schmidt, D.Levine, and C. Cleeland.: Architectures and Patterns for High-performance, Real-time CORBA Object Request Brokers. In Advances in Computers, Marvin Zelkowitz, Ed., Academic Press, 1999.
8. J. Zinky, D. Bakken, and R. Schantz.: Architecture Support for Quality of Service for CORBA Objects. Theory and Practice of Object Systems, vol. 3, no.1, Jan. 1997.
9. M. Satyanarayanan.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications, Vol.8, no.4, pp 10 –17, Aug. 2001.
10. H, Karloff.: Linear Programming. Birkhauser, 1991.
11. Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt.: Gaia: A Middleware Infrastructure to Enable Active Spaces. In IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002
12. Wanghong Yuan, Klara Nahrstedt.: R-EDF: A Reservation-Based EDF Scheduling Algorithm for Multiple Multimedia Task Classes. Proc. of the Seventh Real-Time Technology and Applications Symposium, Taipei, Taiwan, pp. 0149. 2001.

AnyCom: A Component Framework Optimization for Pervasive Computing[1]

Wenzhi Chen, Zhou Jiang, and Zhaohui Wu

Department of Computer Science and Technology, Zhejiang University, China
wzchen@cad.zju.edu.cn, z8j1@zju.edu.cn, wzh@cs.zju.edu.cn

Abstract. Pervasive computing lays emphasis on the seamless integration of human, computers as well as the environment, which challenges the traditional component-based technology. In this paper, after analyzing many previous component models, we discuss a series of problems involved in Pervasive computing: How to administer plenty of embedded computing devices, how to discover components initiatively, how to conquer the heterogeneity of different components, how to implement component cooperation mechanism, and so forth. Finally, we bring forward a new framework AnyCom (a component framework for pervasive computing) and some experiments based on it.

1 Introduction

After a decade of hardware progress, many crucial elements of pervasive computing that were exotic in 1991 now become viable commercial products. Small-embedded systems widely appear in everything, ranging from daily application for mobile telephones and PDA to critical systems for medical diagnostics, automotive electronics, avionics missions, climate control, and manufacturing. Its main task is to engage the physical world to interact directly with sensors and actuators. We characterize a pervasive computing environment as one saturated with computing and communication capability, yet so gracefully integrated with users that it becomes a "technology that disappears." However, besides the problems already identified in its predecessors, namely distributed systems and mobile computing, pervasive computing has some distinct characteristics as follows: effective use of smart spaces，invisibility，context-aware, localized Scalability and masking uneven conditioning [1].

As the complexity of embedded systems increases, pervasive computing requires innovational software architecture. Therefore, applying thinking of middleware and component based development into Pervasive computing software turns to be a

[1] This work was supported in part by the Hi-Tech Research and Development Program of China (863 Program) under Component-based Embedded Operating System and Developing Environment (No.2004AA1Z2050), and Embedded Software Platform for Ethernet Switch (No. 2003AA1Z2160); In part by the Science and Technology Program of Zhejiang province under Novel Distributed and Real-time Embedded Software Platform (No. 2004C21059).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 236-242, 2005.

popular challenging research problem. A software component is defined as a unit of composition with contractually specified interfaces and explicit context dependencies, which can be deployed independently and is subject by third parties. However, when developing a component, designers must consider the following points:

Component size. Components must have proper size to guarantee some quality issues such as correctness, robustness and careful specification.

Component interfaces. To guarantee independence, component should maintain a strict separation between interface specifications and interface implementation.

Component tools and infrastructures. Components must be implemented, assembled, and interact with other components. Therefore, they require tools that may be specialized to component assembly and construction, and also require some basic support structures (infrastructure) providing the means for their interaction.

2 Related Work

We concentrate our attentions on systems that address issues such as configuration and reconfiguration, composition of operating systems, component-based software for embedded operating systems. Most of these systems try to define operating system components. However, they use different design approaches and infrastructures.

Choices and **OS-Kit**[2] address operating system configuration and customization issues as well as component software for operating systems. **Choices** uses a complex object-oriented framework to build a full operating system. **OS-Kit** provides a set of operating system components that can be combined to configure an operating system. However, they don't supply any rules to help build an operating system.

2K cares the adaptability issues to allow applications to be as customizable as possible. **UIC** defines a standard skeleton structure targeted at object-oriented request brokers (CORBA, Java RMI, and DCOM). **GAIAOS**, a middleware infrastructure to enable active spaces, can manage different types of components through UOB (Unified Object Bus). **2K** [3]**, UIC** [4] and **GAIA**[5]**,** all designed by UIUC, have the same weakness: lacking a series of tools to package, install and supervise components.

Recently, the **CCM** [6] extends the CORBA object model by defining features and services that enable application developers to implement, manage, configure, and deploy components that integrate commonly. However, coupling of component meta-data and functionality in CCM is still tight, reducing reusability in some degree.

3 Characteristics of AnyCom

After analyzing these previous systems, we conclude some necessary characteristics required for AnyCom in order to support embedded system in Pervasive Computing.

Administering various devices. The embedded devices in pervasive computing are uncountable, from small to big, from simple to complex. Optimized component framework should be competent enough to administer all the devices.

Discovering components Spontaneously. Pervasive computing environment is dynamic rather than static. New components may come into it, while expired ones will leave away. The process of adding or deleting components should be carried out dynamically at running time, with low (preferably no) human involvement.

Shielding heterogeneity of components. Components designed by different developers probably differ in their granularity, memory size, interfaces, as well as functionalities. How to shield the distinctness among diverse components and exploit them effectively becomes a critical problem in pervasive computing.

Enabling communication in components. Only after organized systematically, can all the distributed components connect and cooperate with one another to perform an actual task. Communications between heterogeneous components are considerable complex, involving authorization and authentication, coding and encoding, etc.

Everything is component. Not only all kinds of services and resources are encapsulated as components, but also the tools and system services that manage other ordinary components are regarded as special kinds of components. All of them are encapsulated as Unified Components, which we will elaborate later in section 5.2.

4 Architecture of AnyCom

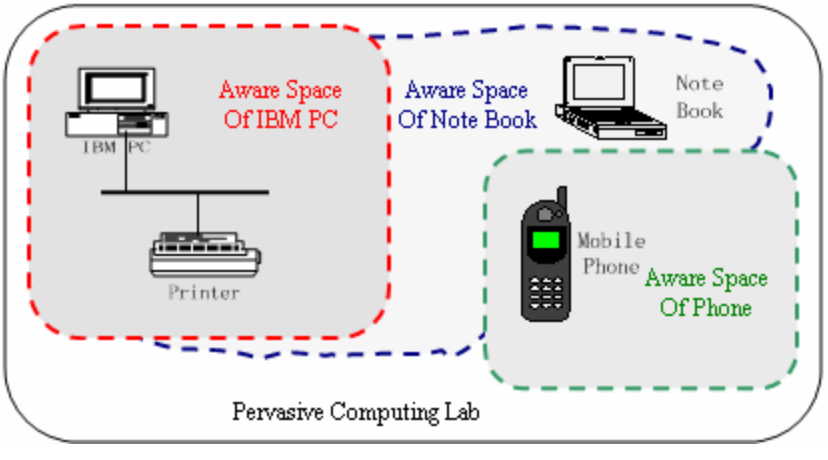


Fig. 1. Example of aware space in Z-CLP

By extending the reach of traditional computing systems to encompass the devices and physical space surrounding the machines, entities, both physical and virtual, may be allowed to seamlessly interact. Compared with the "Active Space" defined as a generic computing system in GAIA [5], "**Aware Space**" is promoted in this paper, referring to the specific physical space, in which one device can be aware of any component, and beyond which the device can not be aware any component. Components in pervasive environment are mobile rather than stationary, so they cannot always be accessed in a fixed location. When a component moves in, it will be added to the aware space; on the other hand, when it moves away, it will be deleted from the aware space. The essence of aware space is the gathering of all components

of a specific device at a specific time. Figure 1 shows the Aware Space in AnyCom. IBM PC, notebook computer, and mobile phone have their aware space respectively, which overlaps in some extent. Therefore these devices can be aware of each other.

4.1 General Architecture

There are three types of elements in the Aware Space: first, the **Task Manager**, responsible for task management; second, the **Component Register**, recording all the available components currently; third, some **unified components,** providing the various abstract services. (Figure 2 shows the general architecture of AnyCom)

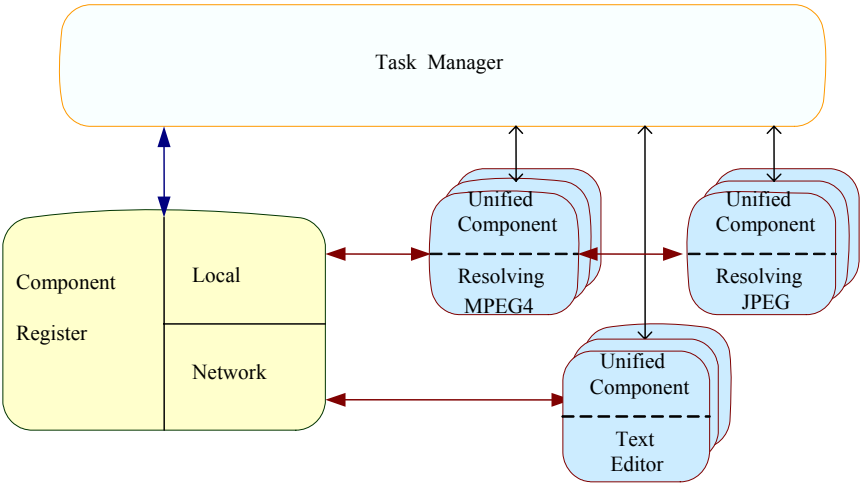


Fig. 2. The general architecture of AnyCom

Task Manager collects information on the physical context and report relevant events in the physical context. It may have different degrees of sophistication in each environment, depending on the sensors deployed in that environment. Examples of sophistication dimensions are user recognition (authentication,) location, activity, and other people in the vicinity.

Task Manager also monitors Quality of Service information provided by the components supporting the user task. Whenever that information becomes incompatible with the requirements of the current task, or the monitored component just dies, Task Manager queries the Component Register to find an alternative configuration to support the task.

Component Register checks the aware space periodically, and keeps up-to-date record of all the available components. Whenever a new component enters aware space, it will be discovered and be recorded in the Component Register; similarly, whenever a component moves away, its record will be deleted from it. Through Component Register, the system can be conscious of how many components are available at any time.

Component Register is divided into two parts: Local Registration to record local components and Network Registration to record components from other devices. Local components are relatively stable and administered by local devices; while network components are liable to change frequently and can only be accessed after authentication. For the sake of security, network components can't be reconfigured or destroyed by local devices.

Unified Component is an elementary capsulated unit, providing the abstract services that tasks are composed of. More detailed concerning Unified Component will be introduced later in section 5.2.

4.2 Unified Component

Aware Spaces are highly heterogeneous by definition. They include a variety of hardware devices and diverse software protocols. However, in order to export a space as a programmable entity, programmers must be offered a common interface to manipulate components, regardless of specific properties and details of the hosting device. Therefore, **Unified Component** is introduced to hide such heterogeneity. Different types of common components (e.g., CORBA, Scripts, and Java Beans) are encapsulated in **Containers**, manipulated by the **Component Managers**, and descried by **Component Control Blocks** to form Unified Component.

A **Container** encapsulates a Unified Component implementation and provides a run-time environment for the component it manages that can:

1. Activate or deactivate component implementations to preserve limited system resources, such as main memory and CPU.
2. Forward client requests such as transaction, security, persistent State, and notification services, thereby freeing clients from having to locate these services.
3. Provide some interfaces of the Unified Component to communicate with each other, such as Facets, allowing component to expose different views to its clients, Receptacles, offering a standard way to specify interfaces required for the component to function correctly, and Event Channel, sending and receiving asynchronous events.
4. Provide some attributes of the Unified Component to be configured by clients or other components when necessary.

A **Component Manager** is responsible for the lifecycle management of a Unified Component, including: creating, deleting and modifying, and also for keeping track of the component after it has been created. Moreover the component manager is a default entry point to the Unified Component. Every component container instantiates default a component manager. This component manager exports the functionality required to manipulate components running in the component container.

Component Control Blocks (CCB) records plenty of meta-data of the Unified Component, such as its ID, its property, its lifetime, its owner, its creating time, its granularity, its function, its interface description, and its memory size. All the meta-data is descried by XML in order to make it universal to most systems. Through CCB, system can get enough information of a specific Unified Component, which helps to perform a proper task.

5 Experiments and Conclusion

Our research work is a part of the National High Technology Development 863 Program of China. The AnyCom architecture has been already applied in some demo systems: "Component-based GUI system", and "Simulated stock system". All these demos are demonstrated in embedded, mobile and distributed computing devices (PowerPC chips and StrongARM XScale chips), showing that the architecture is competent for pervasive environment.

The following figures show the scenario of MMS (Mobile Music Space), in which users are able to enjoy its favorite music without concerning about the location and music transferring between different input devices. When the user who is listening to the music from a mp3 player enters a new place (a vehicle space etc.), the music will automatically be transferred to a more sophisticated device in the new place, and the music will continue to be played.

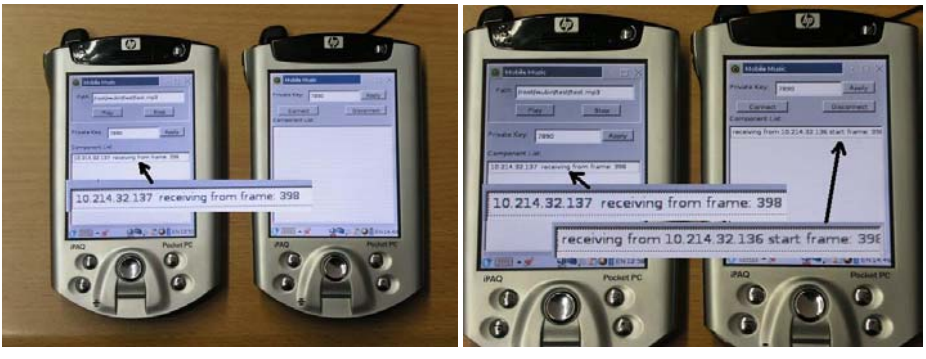


Fig. 3. The scenario of Mobile Music Space

The component framework AnyCom facilitates software development in pervasive computing, by solving problems such as: how to administer plenty of embedded computing devices, how to discover components initiatively, how to conquer the heterogeneity of components and how to implement component cooperation mechanism.

Reference

1. M. Satyanarayanan: "Pervasive Computing: Vision and Challenges", IEEE, Personal Communications, 2001
2. B. Ford et al., "The Flux OSKit: A Substrate for Kernel and Language Research," *Proc. 16th ACM Symp. Operating Systems Principles*, ACM Press, New York, 1997, pp. 38-51.
3. Fabio Kon, Fabio Costa, Roy Campbell, Gordon Blair. The Case for Reflective Middleware, Communications of the ACM. Vol. 45, No. 6, pp. 33-38. June, 2002.
4. Manuel Roman, Fabio Kon and Roy H. Campbell. Reflective Middleware: From Your Desk to Your Hand. IEEE Distributed Systems Online Journal, 2001

5. Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. , In: IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

6. Nanbor Wang, Craig Rodrigues, Chris Gill. A Qos-aware CORBA Component Model for Distributed, Real-time, and Embedded System Development. In: OMG Workshop in Embedded & Real-Time Distributed Object Systems, Object Management Group, 2003.

Association Based Prefetching Algorithm in Mobile Environments

Ho-Sook Kim and Hwan-Seung Yong

Dept. of Computer Science and Engineering, Ewha Womans University,
Daehyun-dong Seodaemun-gu, Seoul, Korea
{khosook,hsyong}@ewha.ac.kr

Abstract. In this paper we propose a prefetching algorithm called STAP (Spatial and Temporal Association based Prefetching algorithm). Our methods are based on the analysis of the spatial and temporal associations of the user's request using data mining techniques. First, we exploit an "associative class set" consisting of an itemset of service classes that is close both spatially and temporally and frequently requested together. With the first method, our prefetching algorithm can select a candidate set that is spatially and temporally associated with the previous request of a user. It is shown that through performance experiments STAP is effective in improving system performance.

1 Introduction

With the development of wireless data communication and portable information devices, mobile computing is becoming very popular. Especially, location-based services(LBS) such as map service information or PDA based web surfing have expanded rapidly. However, information services in the mobile hosts are limited to the network resources provided by the wireless communication environment. Data caching and prefetching on mobile hosts have been considered effective solutions to improve mobile communication services and a plethora of techniques have been proposed in literature to handle these issues [1,2,3]. But in real life, mobile equipments have a small cache compared with a server's spatial database, which leads us to consider another criteria for precise prefetching on LBS.

In this paper, we propose a new prefetching algorithm, STAP (Spatial and Temporal Association based Prefetching algorithm). To support LBS applications, STAP considers the spatial and temporal association among service classes in users' queries as well as the objects' weights and user's access pattern. STAP uses an "associative class set" to decide which objects are prefetched. The associative class set is composed of a frequent itemset that is close both spatially and temporally and requested frequently together. A frequent itemset is generated using mining techniques such as clustering and the association rule. Through performance experiments, we prove the efficiency of the proposed STAP by comparing it with a group of existing cache management algorithms.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 243–250, 2005.

Table 1. Access records of LBS

User ID	User Position	Class	Object ID	Object Position	Access Time
A	(323,334)	Hospital	98	(432,400)	17:03:02
A	(340,337)	Terminal	54	(200,189)	17:04:35
A	(355,339)	Pharmacy	74	(450,388)	17:05:50
C	(389,273)	Hotel	77	(400,250)	20:04:23
C	(402,250)	Restaurant	63	(410,265)	20:07:40

2 Generation of the Associative Class Set

Table 1 shows the server's access record format used in LBS. In Table 1, User "A" accesses a special hospital that is far from him and then requires a pharmacy near the hospital in a few minutes. User "C" requires the hotel that is nearest to his current position and then requires a restaurant near the hotel. Like these examples, we are able to assume that there is a relation between the queries required by the same user in a short time. We also assume that the objects required by the related queries are located in close proximity to each other in real space. In a location-based application, if a user finds a hospital and then requires a pharmacy in a second, then we may assume that the pharmacy has to be near the previous required hospital. In this case, the query type must be as follows: "Find a pharmacy which is located within 100 meters from the hospital X". In Table 1, User "A" accesses a terminal and a pharmacy in a short time after requesting the hospital. The location of the terminal is, however, far from the hospital and the pharmacy, so we can determine that there is a spatial association only between the hospital and the pharmacy. As the result, we can classify the set of objects as two groups {98, 74} and {77, 63} that have both the spatial and temporal relations simultaneously. The goal of our research is to generate the associative class set based on the set of service classes such as {hospital, pharmacy} and {hotel, restaurant} by association rules.

Figure 1 depicts the steps to generate the associative class set. First, we make a user sequence that is the sorted list of each user's access records by access time's order. And then we generate temporal transactions. Temporal transaction is a group of requests where the time delay between requests is less than a certain threshold such as the time window of [4]. A great deal of research [4,5] has been conducted to resolve that problem. So we will skip the detailed explanation of the generation algorithm of the temporal transaction. Next, we group objects in each temporal transaction into several TSNO (temporal and spatial neighbor objects) so that objects in each TSNO may be close to each other temporally and spatially. In this step, we propose two clustering algorithms according to the application purpose. The algorithms will be described in Section 2.1. in detail. What we try to generate is not the associative object set but the associative class set so that, we replace each object in TSNO with its class attributes. Finally, we generate the associative class set as a meaningful frequent itemset considering both temporal and spatial characteristics in mobile environments.

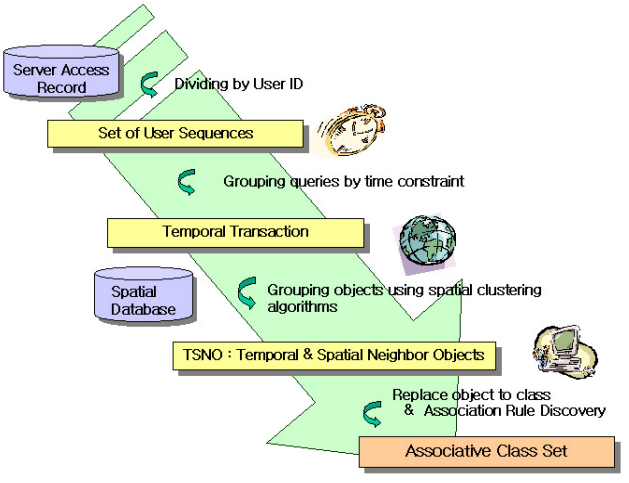


Fig. 1. Generation steps of associative class set

Many association rule generation algorithms have been developed, and we use PolyAnalysis 4.5 in [7] to generate the frequent itemset and association rule.

2.1 Discrimination of TSNO by Spatial Constraints

In this step, a variety of clustering algorithms may be applied according to the purpose of their applications. Many researchers have investigated clustering algorithms. Among of them, we utilizes two clustering algorithms suitable to extract the business relations existed between spatial object's classes.

Clustering Algorithm by Maximum Distance (CAMD): In CAMD, all related objects are included in a circle of diameter r_1. Here, r_1 is a maximum distance as the limitation of spatial constraints. For example, If Tom wants to find a suitable spot to open a new restaurant. For this, the candidate area may include many objects such as office buildings and shopping complexes having a relationship with a restaurant within a constant distance. In this case, the spatially related area may have a limitation that all objects are located within a specific distance and may have a convex polygonal shape.

Clustering Algorithm by Connected Objects (CACO): In the CACO algorithm, all related objects are defined as a connected neighbor set. It means that every element of TSNO has at least one neighbor within distance r_2. Here, r_2 is a distance as a limitation of spatial constraints and the length of r_2 is generally much smaller than r_1 in CAMD. For example, the distribution of stores according to a street or, the distribution of houses around a lake is made by a set of long connected spatial objects in the shape of a snake.

3 Association Based Prefetching Algorithm

The process of cache management is summarized as follows. When a user tries to get the information of object a_1, the mobile host checks to see if a_1 resides in its cache. If the cache doesn't have a_1, the mobile host requests object a_1 to the server. In the server, when a_1 is requested, a prefetching mechanism decides which objects are selected as candidate objects with high access probability, and then the server sends object a_1 and the recommended objects to the mobile host. In the mobile host, the cache replaces the victim objects to make room for newly prefetched objects.

Input: requested spatial data from a mobile host (a_1), maximum distance
Output : recommendation object set
{
A = the class attribute of a_1 ; // **step 1**
S = associative class sets which have the class A as one of elements; // **step 2**
C = all classes which are included in S; // **step 3**
Candidate objects = select all objects which are dist(object, a_1) < maximum distance and whose class attributes are included in C; // **step 4**
Calculate the prefetching score of object included in candidate objects; // **step 5**
Select recommendation object set from candidate objects by descending order of prefetching score;
Recommendation object set = recommendation object set + landmark objects those are near by a_1; // **step 6**
}

Fig. 2. Spatial and Temporal Association based Prefetching algorithm

In this section, we propose a new prefetching algorithm STAP and compare with SP. SP(Spatial property based Prefetching algorithm) considers user's direction and the distance between a user and an object when it recommends candidate objects in the server. SP selects the closest n objects from the user, which are located on the same direction as the user's movement. Direction and distance are commonly used criteria in cache management [1,2]. STAP, our proposed algorithm, considers both temporal and spatial associations included in the user's consecutive queries, landmark information, as well as the weight attribute of objects. Figure 2 shows how STAP works. When object a_1 is requested to the server, STAP selects A as the class attribute of a_1. Then it finds associative class sets which include the class A. In step 3, STAP selects all classes which are included in the same associative class set with class A. In step 4, it chooses candidate objects, which are close to a_1 and whose class attribute is one of the classes generated in step 3. And then, it calculates the prefetching score of each element in candidate objects. In step 6, we add the landmark objects close to a_1 to the recommendation object set. According to [10], when a user tries to get geographic information about a particular area, he/she tends to require a landmark object such as a school or an administrative office that draws attention to that area, and this query pattern can occur in many kinds of LBS. Finally, the recommendation object set and a_1 are sent to the mobile host. The prefetching score used in STAP is calculated as in Equation (1).

$$Prefetching\ score\,(a_1, o_1) = association\ score + weight\ score \quad (1)$$

The association score of a_1 and o_1 refers to how deeply class O (object o_1's class) is related to class A (object a_1's class). We assign the association score of an object based on the support value of each associative class set. The weight score of o_1 may have different implications according to the applications. In many cache algorithms, the access count of the object is used as the weight [2]. In [3], the weight is the Voronoi area based on the location of the other object with the same class attribute.

4 Experiments

In this section, we implement the proposed algorithms and performed several experiments to evaluate their performance. Experiments 1 and 2 show the changes of the discriminated TSNO number according to the increase of maximum distance using COMD and CACO algorithms, respectively. We may estimate the appropriate maximum distance value according to their results. Experiments 3 shows the performance of STAP.

4.1 Experimental Environments

In our experiments, we used a total of 50,000 objects. The total class number reaches 500, and each class includes 50 to 150 objects. In the test area, 200 landmark points are randomly distributed. The weight of each object is assigned randomly from 1 to 20. We subdivide the entire test area into 10 sub-areas of different size and each sub-area includes objects of 5% to 20% of the total spatial data. These data points are distributed randomly in the sub-area with Gaussian distribution. The total number of queries used in our test is 90,735, and the total number of users is 100.

4.2 Experiment Results

We generate the associative class set using 60% of queries in the query table, and as a result, 77 associative class sets are generated when the support was 0.01%. Each associative class set has 2 to 5 class items. And then we evaluates the performance of STAP using the rest 40 % of queries. As shown in the graphs, we compare two prefetching algorithms, STAP and SP, using two different cache replacement algorithms: LRU and FAR. LRU selects a victim for the least recently used object in its cache. FAR (Furthest Away Replacement) in [1] is the representative semantic based cache replacement algorithm in a spatial database. It considers the moving direction of a mobile user, its speed, and the distance from a user to objects when it selects the victim.

Experiment 1: Discriminated TSNO number with different maximum distance using CAMD algorithm We have tested the number of

discriminated TSNO as the maximum distance increased from 60 to 100 by applying CAMD algorithm. We prune the single-TSNO that includes only one object (they have no influence on the generate association rule). As a result, the number of discriminated TSNO decreases as the maximum distance increases. When the maximum distance is too small, the number of single-TSNO increases and then many single-TSNO have to be pruned. The result is shown in Figure 3(a).

Experiment 2: Discriminated TSNO number with different maximum distance using the CACO algorithm We have tested the number of discriminated TSNO as the maximum distance increased from 10 to 50 by using the CACO algorithm. As a result, the number of discriminated TSNO decreases as the maximum distance increases. However, when the maximum distance is very small, the number of the single-TSNO increases so a large number of TSNO are pruned. When the value of the maximum distance is more than 40, the number of the last generated TSNO decreases. Moreover, we have known that there is scarcely any change in the number of the last generated TSNO when the maximum distance is more than 30 as shown in Figure 3(b). In this case, the maximum distance value 30 is the meaningful spatial constraint to generate TSNO. The result is shown in Figure 3(b).

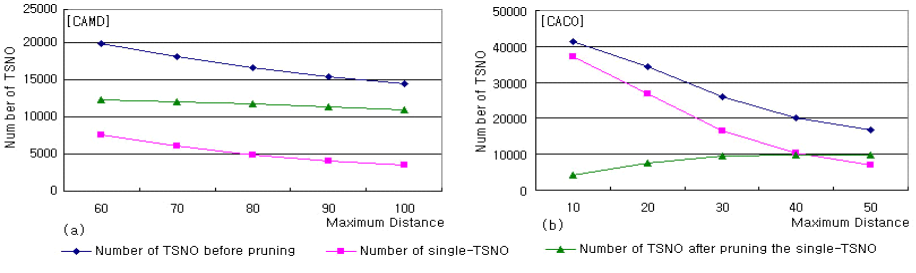


Fig. 3. TSNO number according to increase of maximum distance using CAMD and CACO algorithm

Experiment 3: Cache hit ratios with different cache sizes We compare the performance of three prefetching algorithms and two cache replacement algorithms, increasing the cache size from 0.5% to 5% of the server's database size when the prefetch size is 2% of the mobile host's cache size. Figure 4 shows how the cache hit ratio increases as the cache size increases. As shown in the graphs, the STAP method shows much better performance than both the SP and No-Prefetch; moreover, it is known that when the same prefetching algorithm is used, FAR has a better hit ratio than LRU.

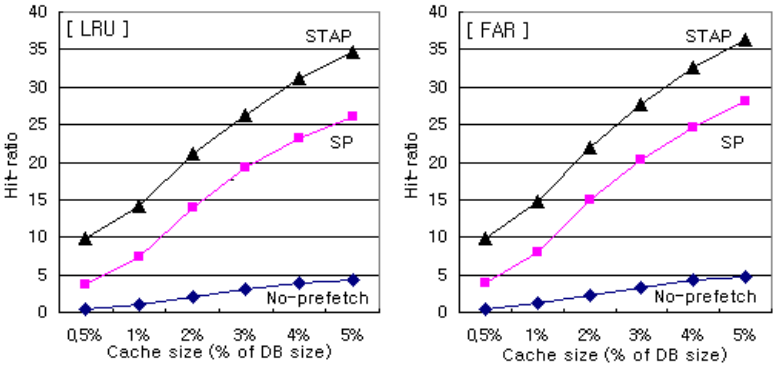


Fig. 4. Cache hit ratio of algorithms as the cache size varies from 0.5% to 5%

5 Conclusion

In this paper, we have proposed a new association-based prefetching algorithm STAP that efficiently supports location-based services in mobile environments. STAP considers the spatial and temporal association of the requested class and the spatial data's weight. To take both spatial and temporal associations of classes into consideration, STAP exploits an "associative class set", consisting of an itemset of service classes that is close both spatially and temporally and are frequently requested together. The experimental results show that STAP outperforms the SP by considering only the user's location, direction, and distance.

References

1. Qun Ren and Margaret H. Dunham, "Using Semantic Caching to Manage Location Dependent Data Mobile Computing", *Proceedings of MobiCom 2000, Boston, Massachusetts*, pp. 210-221 (2000).
2. Uwe Kubach and Kurt Rothemel, "Exploiting Location Information for Infostation-Based Hoarding", *International Conference on Mobile Computing and Networking*, pp. 15-27 (2001).
3. Baihua Zheng, Jianliang Xu, and Dik L.Lee, "Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environment", *IEEE Transactions on Computers* , **51**(10), pp. 1141-1153 (2002).
4. Rakesh Agrawal and Ramakrishnan Srikant, "Mining Sequential Patterns", *Proceedings of International Conference on Data Engineering* , pp. 3-14 (1995).
5. Rakesh Agrawal, Tomasz Imielinski, and Arun Sqami, "Mining Association Rules Between Sets of Items in Large Databases", *Proceedings of the International Conference on Management of Data (SIGMOD)*, pp. 207-216 (1993).
6. Yasuhiko Morimoto, "Mining Frequent Neighboring Class Sets in Spatial Databases", *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* , pp. 353-358 (2001).
7. PolyAnalyst 4.5, "http://www.megaputer.com.".

8. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, "CURE: An Efficient Clustering Algorithm for Large Databases", *Proc. of the ACM SIGMOD Conference on Management of Data*, pp. 73-84 (1998).
9. Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", *Proc. of ACM SIGMOD 3rd* , pp. 226-231 (1996).
10. Y.-D.Seo, K.-H.Ahn, and B.-H.Hong, "Analysis of Using Internet GIS", *Korea Information Science Society SIGDB* , **18**(1), pp. 41-52 (2002).
11. Dhananjay S. Phatak and Rory Mulvaney, "Clustering for Personalized Mobile Web Usage", *Proceedings of the IEEE FUZZ'02* , pp. 705-710 (2002)

Integration Policy in Real-Time Embedded System

Hyun Chang Lee

School of Information and Technology, Hansei University, 435-742
604-5, Dangjung-Dong, Gunpo-Si, Kyunggi-Do, Korea
hclglory@yahoo.co.kr
http://www.hansei.ac.kr

Abstract. Interoperability of data between wire and wireless environment makes it easier for embedded and enterprise systems to interact with each other. They can exchange data, load and run application objects. These activities are getting increasingly in demand where embedded systems interact with the enterprise world. Also, the availability of technologies that enable mobile access to data has brought great expectations that users would be able to access information any time, anywhere within embedded system environment. However, the existing infrastructure and content are not well-suited for other types of accesses. In this paper, we present an incremental and efficient integration and maintenance algorithm for analytical purposes by integrating updates generated from embedded systems through a wireless PDA or smart phone, or hands-free access. The proposed algorithm can also reduce the overhead in managing queries for integration and maintenance.

1 Introduction

The explosion in the use and availability of accessing the analysis data through embedded systems anytime and anywhere has great fascinating prospect. However, the existing infrastructure and content are not well-suited for other types of accesses, e.g., devices that have less processing power and memory, small screens, and limited input facilities, or through wireless data networks with low bandwidth. Thus, there is a growing need for techniques that provide alternative means to access analysis content and services. To access analytical information, we often refer to data warehouse.

Data warehouse is very large databases which contain historical, summarized, integrated and time variant collection of data from information sources with embedded systems and integrates it into a single source where it can be queried by clients of the warehouse. In short, a data warehouse is a repository of integrated information, available for queries and analysis (e.g., decision support, data mining)[4]. When relevant information is modified by clients using embedded system like wireless PDA or smart phone etc., the information is extracted from the embedded data source, translated into a common model, and integrated with existing data at the warehouse. Queries can be `replied` and analysis can be performed quickly and efficiently since the integrated information is available at the warehouse.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 251-257, 2005.

The relations stored at the warehouse represent materialized views over the data source [2]. Because queries at the warehouse tend to be long and complex, a warehouse may contain materialized views in order to speed up query processing [3]. As changes through embedded systems are made to the data at the source, the views at the warehouse become out of date. In order to make the views consistent again with the source data, changes to the source data in commercial warehousing systems are queued and propagated periodically to the warehouse view in a large batch update transaction. An important problem in data warehousing is how to execute queries and the periodic maintenance transaction so that they do not block one another, since both queries and maintenance transactions can be long running [5].

One of various approaches is to allow users to see an inconsistent database state. However, such inconsistency is not acceptable. During analysis it would be unacceptable to have the results change from query to query. Therefore, most of the research on materialized views has focused on techniques for incrementally updating materialized views in order to make the views consistent, and numerous methods have been developed for materialized view maintenance [1]. Some representative approaches for data integration and maintenance are the eager compensating algorithm, ECA, and recomputing view, RV, in centralized environment [7].

In this paper, we propose an incrementally efficient data integration algorithm in embedded system environment in order to solve or reduce the overhead in managing queries at warehouse. Our algorithm could also reduce server loads by transferring the answer that would be compensated at the data warehouse to each data source. The paper is organized as follows. In section 2, we briefly review related researches, and in section 3, we provide data integration model. In section 4, we propose TCache algorithm. Section 5 describes the performance of the proposed algorithm. In section 6, we conclude and discuss future directions of our work.

2 Related Research

To integrate updated data through real-time embedded environment, warehouse collects information from one or more data sources using embedded systems and integrates it into a single database, a large repository for analytical data and decision support, where it can be queried by clients of the warehouse. A general description of data warehousing may be found in [4] and the main problem that arises in the context of a data warehouse is to maintain the materialized view at the data warehouse in the presence of updates through embedded system like wireless PDA to the data sources.

When the simple update information occurring from source using embedded system arrives at the warehouse, we may discover that some additional source data is necessary to update the view. Most of the incremental algorithms to integrate and maintenance data focus on immediate update, which updates the view after each base relation is updated as illustrated in figure 1[1,6].

Some algorithms [7,8] are based on deferred update, which updates the view only when a query is issued against the view. A simple approach of one algorithm in centralized environment is recomputing view, RV, algorithm. In RV, data warehouse can recompute the whole view periodically, and the time to begin recomputing the whole

view is whenever update occurs at the data source. The major drawback in RV is excessive consumption of resources and time.

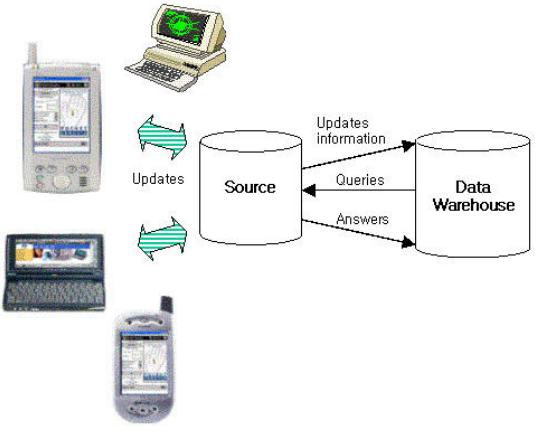


Fig. 1. Query Processing model through embedded systems

A more appropriate solution would be to update the data warehouse incrementally in response to the updates arriving from data source through embedded systems. Eager compensating algorithm, ECA, is an immediate and incremental integration algorithm that avoids the overhead for recomputing whole view at a time. The algorithm is applied to a centralized environment and contains the following problems. First, data warehouse after receiving update information from embedded data source has to manage the unanswered query set until all of answers arrive from embedded data source. Second, all updates get to be transmitted from data source to the warehouse without only information associated with the view. This causes an increase in the amount and number of messages transferred between data source and warehouse. To solve or reduce these problems, we propose an incremental integration and maintenance temporary cache algorithm called TCache.

3 Model for Integration Policy

First, we mention the data model to be managed at the data warehouse environment. The data model in this paper is based on [6]. Updates occurring at the data sources are classified into three categories. Type one is a single update transaction where each update is executed at a single data source. Type two is a source local transaction where sequences of updates are performed as a single transaction. However, all of the updates are directed to a single data source. Type three is a global transaction where the updates involve multiple data sources.

For the purpose of this paper, we assume that the updates being handled at the data warehouse are of type one and two. At each source, there are five types of message from the data source to the warehouse. One is reporting the update information and maintaining the reported update in a temporary storage to keep the order of update

occurring at the source after receiving the update information from embedded systems. We call the storage temporary cache, TCache.

The next step is that source determines whether or not the update information is related to the tables associated with the view after receiving the view definition. If it is not related, then process executing step. Otherwise just process next examining step. The third step is examining the TCache storage and evaluating. Next step is just executing the current the update information and as a last step, it is sending the answer result corresponding to the update to the warehouse.

Figure 2 shows the structure of the data warehouse environment for the data warehouse and sources described above. It consists of n distributed sites for data sources and another site for storing and maintaining the materialized view of data warehouse. We assume that messages between each data source and data warehouse are not lost and are delivered in the order in which they are sent. The database model for each data source is assumed to be a relational data model. A data source may store any number of base relations. Updates are executed atomically and transmitted asynchronously to the data warehouse as updates occur.

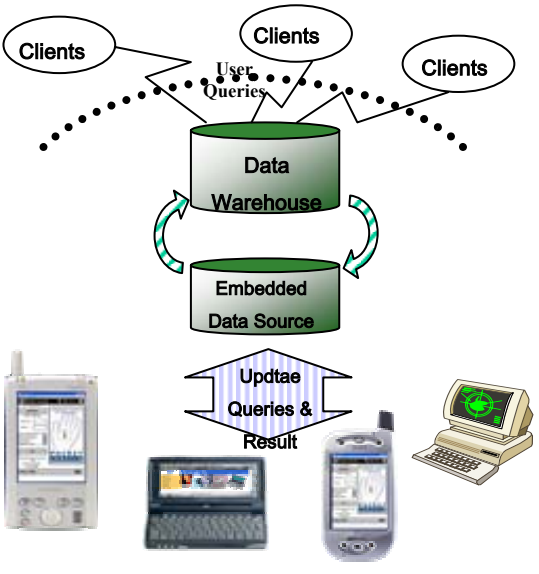


Fig. 2. Data integration model at embedded system environment

4 Methodology of TCache

In this section, we describe the data integration and maintenance algorithm called TCache. The basic steps to integrate data into warehouse are as follows. A notification for update occurring from embedded system is sent to the warehouse to only get a view definition. The warehouse receiving the request from the embedded data source sends back the view definition to the data source. After receiving the view

definition, the source evaluates the query to determine whether or not it is related to tables associated with the view. The answer corresponding to the query is generated and sent to the warehouse. Warehouse receiving the answer from the data source applies it to the materialized view to maintain a valid state for the updated view.

To explain the events that occur during data integration and maintenance, the events that occur in the data source and the warehouse view are identified. The events at the data source are ES_req, ES_chk, ES_eva, ES_exe and ES_sed while the events at the warehouse are W_vie and W_ans.

1) Events at the embedded data source

- ES_req : the embedded data source enrolls the update information into TCache and sends the update notification to the warehouse.
- ES_chk : the embedded data source determines whether or not the update is related to the tables associated with the view after receiving view definition from warehouse. If it is not related, then process ES_exe, else next step.
- ES_eva : the embedded data source examines the TCache to determine whether or not there is any update among previous updates in TCache. If there is, then wait, else process ES_exe step.
- ES_exe : the embedded data source executes the update and deletes the current update in the TCache.
- ES_sed : the embedded data source sends the generated answer after evaluating.

The events, ES_req, ES_chk, ES_eva, ES_exe and ES_sed can be processed separately.

2) Events at the data warehouse

Suppose that the events of the data warehouse are processed in the following orders of a transaction occurred from embedded system.

- W_vie : the warehouse receives an update U and sends the view definition corresponding to the update to the embedded data source.
- W_ans : the warehouse receives an answer relation A from embedded data source and immediately updates warehouse.

We will assume that events are atomic, and actions within an event follow the order described above. For example, within an event ES_req, the embedded data source first records the update information in TCache and sends the update notification to the warehouse.

5 Performance Evaluation

In the previous sections, we mentioned several strategies for integration policy in a data warehouse environment. In this section, we analyze the performance of the maintenance algorithm to integrate data in embedded system environment. We will briefly compare it with the Strobe and Sweep algorithms including ECA. The main properties of TCache are that it reduces the amount of messages transferred between the warehouse and embedded data sources, and also reduces server loads by transferring the answer that would be compensated at the warehouse to each source.

The eager compensating algorithm requires that the data warehouse be in a quiescent state for incorporating the new views, whereas there is no such problem in the proposed algorithm. It also increases the warehouse overhead because the data ware-

house processes the operation to compensate temporary results generated that are sent from each source. In addition it increases the response time because a subsequent update has to wait until the preceding update is finished. In TCache algorithm, the previous problems mentioned are reduced or solved because TCache algorithm handles an update as a local compensation at each source. As a result, it can reduce the server loads.

The main properties of TCache are that it reduces the amount of messages and the number of messages transferred between the warehouse and embedded data source, and also reduces server loads by transferring the answer that would be compensated at the warehouse to embedded data source using TCache.

The method used in recomputing view is that warehouse can either recomputed the full view whenever an update occurs at the data source, or it can recompute the view periodically. We expect that the ECA is more efficient than the recomputing view method because it is an incremental data integration and maintenance algorithm. The ECA, however, needs more queries to maintain the view compare to recomputing view method. ECA uses non-blocking method at the data source as mentioned in [1] while TCache uses an optimistic blocking method. At the warehouse, ECA must wait until all answers from the embedded data source arrive at warehouse while TCache applies answers immediately without waiting. Figure 3 shows the performance result of the number of bytes transferred as function of number of updates.

In figure 3, we omit the performance evaluation of worst case because there is a wide difference comparing its performance with others. From figure 3, we see that TCache algorithm shows enhanced result by reducing message transferred between embedded data source and warehouse.

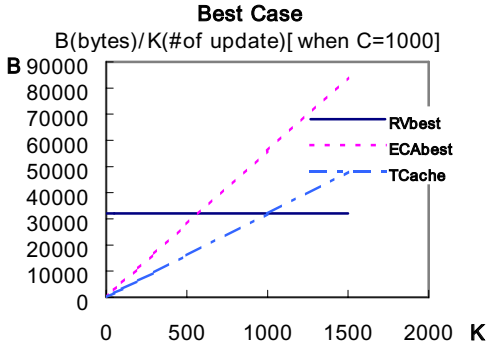


Fig. 3. Transferred Bytes (B) versus number of updates (K)

6 Conclusion

The ability to access the analysis data through embedded systems is a fascinating prospect. Thus, there is a growing need for techniques that provide alternative means

to access analysis content and services. To access analytical information, we often refer to data warehouse. A data warehouse is a subject oriented, integrated, non-volatile and time variant collection of data from information sources with embedded systems and integrates it into a single source where it can be queried by clients of the warehouse. When relevant information is modified by clients using embedded system like wireless PDA or smart phone etc., the information is extracted from the embedded data source and integrated with existing data at the warehouse. Queries can be answered, and analysis can be performed quickly and efficiently since the integrated information is available at the warehouse. In order to make the views consistent with the source data, changes to the source data are propagated incrementally to the warehouse view.

Therefore, in this paper, we propose a new algorithm, TCache, to reduce the overhead in managing queries at warehouse and the amount of messages. We also present brief performance results for the TCache algorithm. The TCache algorithm has the following advantages. It reduces the amount of messages and the number of messages transferred between warehouse and embedded data source. The algorithm also applies answers immediately without waiting. In the future, a more detailed performance evaluation and analysis must be done to evaluate the effectiveness of the proposed algorithm.

References

1. T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In M. Carey and D. Schneider, editors, Proceedings of ACM SIGMOD 1995 International Conference on Management of Data, pages 328-339, San Jose, CA, May 23-25 1995
2. D. Lomet and J. Widom, editors. Special Issue on Materialized Views and Data Warehousing, IEEE Data Engineering Bulletin 18(2), June 1995.
3. V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In Proceedings of ACM SIGMOD 1996 International Conference on Management of Data, pages 205-216, 1996.
4. W.H. Inmon and C. Kelley. Rdb/VMS:Developing the Data Warehouse. QED Publishing Group, Boston, London, Toronto, 1993.
5. D. Quass and J. Widom. On-Line Warehouse View Maintenance. In Proceedings of ACM SIGMOD 1997 International Conference on Management of Data, pages 393-404, 1997.
6. Yue Zhuge, Hector Garcia-Molina, and Janet L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In Proceedings of the International Conference on Parallel and Distributed Information Systems, December 1996.
7. D. Agrawal, A. El Abbadi, A. Singh and T. Yurek. Efficient View Maintenance at Data Warehouses. In Proceedings of ACM SIGMOD 1997 International Conference on Management of Data, pages 417-427, 1997.
8. L. Colby, T. Griffin, L. Libkin, I. Mumick, and H. Trickey. Algorithms for deferred view maintenance. In SIGMOD, pages 469-480, June 1996.
9. A. Gupta and I. Mumick. Maintenance of materialized views: Problems, techniques, and applications. IEEE Data Engineering Bulletin, 18(2):3-18, June 1995.

Prism-WM Based Connector Interaction for Middleware Systems

Hwa-Young Jeong[1], Young-Jae Song[2]

[1] Faculty of General Education, Kyunghee University, 130-701, 1, Hoegi-dong, Dongdaemun-gu, Seoul, Korea
hy_0917@yahoo.co.kr
[2] Department of Computer Science, Kyunghee University, 449-701 Yong-In City Kyungkido, Korea
yjsong@khu.ac.kr

Abstract. A recent emergence of small, resource-constrained, and highly-mobile computing platforms presents numerous new challenges for software developers. Software developers use software architecture method for development in this setting efficiently. Architectural styles represent composition patterns and constraints at the software architectural level and are targeted at families of systems with shared characteristics. We refer to development in this new setting as programming-in-the-small-and-many (Prism). This paper provides a description and evaluation of connector's interaction between connected software components interface, a middleware platform intending to support software architecture-based development in the Prism setting. Also, we consider the characteristics of applied components in this connector, including the bean request processing time and the memory use rate. Results of this research show that components with small memory are given process preference and the processing waiting time of component has been improved.

1 Introduction

In recent years, there has been a increasing trend of migration from the traditional, desktop setting to highly distributed, mobile, possibly embedded and pervasive computing environments. Such environments present daunting technical challenges: effective understanding of existing or prospective software configurations; rapid composability and dynamic reconfigurability of software; mobility of hardware, data, and code; scalability to large amounts of data, numbers of data types, and numbers of devices; and heterogeneity of the software executing on each device and across devices. Increasingly systems are implemented as compositions of independently-developed components that must be integrated into working systems using various interaction mechanisms, such as composing with one another, modifying, and maintaining [4]. Thus, software engineering researchers and practitioners have successfully dealt with the increasing complexity of systems by employing the principles of software architecture. Software architecture provides design-level models and guidelines for composing the structure, behavior, and key properties of a software system.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 258-265, 2005.

An architecture is described in terms of software components (computational elements) [1], software connectors (interaction elements) [2], and their configurations (also referred to as topologies) [3]. An architectural style codifies architectural composition guidelines that are likely to result in software systems with certain desired properties.

In this paper, we implement connector using Prism-MW [5,6]. Prism-MW provides highly efficient and scalable implementation-level support for the key aspects of Prism application architectures, including their architectural styles. We say that the middleware is architectural because it provides programming language-level constructs for implementing software architecture-level concepts such as component, connector, configuration, and event. Thus, we propose the connector that supports the service by the efficiency characteristic to handle the multiplex asynchronous requests of component. The request process of component in connector follows the priority by the efficiency characteristic to efficiently handle and operate components that request service. The priority is set up by bean request processing time and memory use rate. The component is applied in this suggested technique based on EJB, the server side component model. Results from composition and operation of sample EJB present the decrease in average waiting time of components waiting in connector and the increase of count of processed component per unit time.

2 Related Research

2.1 Middleware Technology

Middleware is a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system[9]. Recent improvements in middleware technology and various standardization efforts, as well as market and economical forces, have resulted in a multitude of middleware stacks, such as those shown in Figure 1. This heterogeneity often makes it hard, however, to identify the right middleware for a given application domain. Moreover, there exist limitations on how much application code can be factored out as reusable patterns and components in various layers for each middleware stack[10].

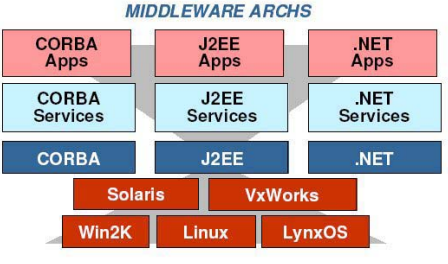


Fig. 1. Multiple Middleware Stacks

There are a small number of different kinds of middleware that have been developed. These vary in terms of the programming abstractions they provide and the kinds of heterogeneity they provide beyond network and hardware [9]. That is Distributed Tuples, Remote Procedure Call, Message-Oriented Middleware and Distributed Object Middleware. Connector can represent various interactions from simple interaction mechanism like procedure call to complicate interaction, client-server protocol and database access protocol [7]. In the client-server style through the synchronous connector like RPC, component communication uses request and response event. Connector in charge of interaction is positioned between the client component and the server component. As the interaction between components, the client component sends the request for service processing to a connector and the connector relays it to the connected server component [8].

2.2 Prism-WM Architecture

Prism-MW is a middleware platform that enables efficient implementation, deployment, and execution of distributed software systems in terms of their architectural elements: components, connectors, configurations, and events. It also provides extensible support for both monitoring and redeployment of resources at the architectural level [11,12]. Figure 2 shows the simple architecture of Prism-MW [5,6].

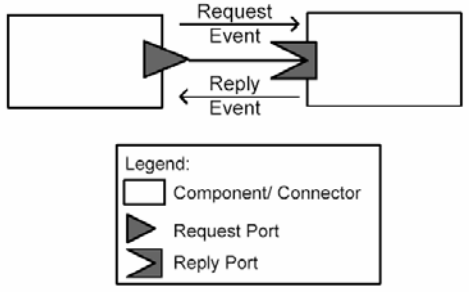


Fig. 2. Link between two ports in Prism-MW. This shows a figure consisting of composition with component and connector. Request events are forwarded from request ports to reply ports, while reply events are forwarded in the opposite direction.

And it should impose minimal overhead on an application's execution and should be scalable in order to effectively manage the large number of devices, execution threads, components, connectors, and communication events present in Prism systems. Prism-MW should be extensible and configurable in order to accommodate many varying development concerns across the heterogeneous Prism domain. These include multiple architectural styles, as well as awareness, mobility, dynamic reconfigurability, security, real-time support, and delivery Guarantees.

3 Prism-MW Based Connector Interaction

3.1 Connector Interaction by Priority Process

In this research, the connector is based on Prism-MW which has simple connecting structure is designed to be applied to the priority considering the efficiency among component specifications. Bean request processing time and memory use rate are selected as the efficiency characteristic factors for this research. For the priority process calculation, the range of selected efficiency characteristic value is decided in proportion to its value and the weight decision table for calculating the weight is composed. Therefore, the sum of each weight represents the relative value of the efficiency characteristic value of request component and the component with small weight sum is handled and performed first. And if the weight sums of request components are the same, we handle them with FIFO method. Table 1 shows the weight (W11 ~ Wn2) by weight decision table on the basis of the efficiency value (R11 ~ Rn2) of applied component and for this, we set up the weight according to the range of each efficiency value as Table 2.

Table 1. Efficiency and weight of component. Rnm : Efficiency value of relevant component, Wnm : Weight, (m:1≤m≤2).

Component efficiency	Request						
	1		2		…..	n	
Bean request processing time	R_{11}	W_{11}	R_{21}	W_{21}	…..	R_{n1}	W_{n1}
Memory use rate	R_{12}	W_{12}	R_{22}	W_{22}	…..	R_{n2}	W_{n2}

Table 2. Weight decision table according to the range

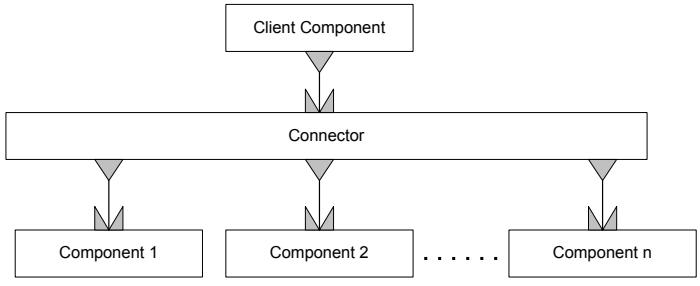
R_{nm} : Range (%)	W_{nm} : Weight
$0 < R_{nm} \leq 25$	1
$26 < R_{nm} \leq 50$	2
$51 < R_{nm} \leq 75$	3
$76 < R_{nm} \leq 100$	4

To decide the weight, the range of practical efficiency value Rnm is calculated by the relative calculation. The sum of weight calculated by weight decision table is the priority estimation value (Si) of relevant component and the priority is decided by this value. Therefore, Si decides the final processing order calculated by adding up weights of each efficiency factors.

$$S_i = \sum_{j=1}^{2} W_{ij}, (1 \leq i \leq n) \quad (1)$$

3.2 Prism-MW Based Priority Connector Interaction

Prism-MW's connector has been implemented in Java. Figure 3 shows the Prism-MW based component composition using connector. In the connector, component process order is calculated by the priority algorithm considering component characteristics.




```
class PriorityPrimMW {
  static public void main(String argv[]) {
    Architecture arch = new Architecture
       ("PriorityTest");
    AbstractImplementation ClientcomponentImpl = new
       ClientcomponentImpl ();
    Component Clientcomponent = new Component("cp",
       ClientcomponentImpl);
          :   :
    AbstractImplementation ComponentnImpl = new
       ComponentnImpl();
    Component Componentn = new Component("da",
       ComponentnImpl);
    Connector conn = new Connector("Conn");
           :   :
    Port portcpReq=new Port(REQUEST); //create port
    Port portConnReq1=new Port(REQUEST);
    Port portConnReq2=new Port(REQUEST);
    Port portConnRepl=new Port(REPLY);
    Port portCom1Repl=new Port(REPLY);
    Port portComnRepl=new Port(REPLY);
    Clientcomponent.addPort(portcpReq);
    Component1.addPort(portcom1Rep);
    Componentn.addPort(portcomnRep);
          :  :
    arch.weld(portcpReq, portComnRep);
    arch.weld(portConnReq1, portCom1Rep);
    arch.weld(portConnReq2, portComnRep);
    arch.start();
  }
}
```

Fig. 3. Prism-MW based component composition

Figure 4 shows the priority process in connector using Prism-MW.

```
public class ConnectorThread extends Connector imple-
ments Runnable {
            :    :
  public void run( ) {
    CalculateWeight( ); //calculate weight for decision
                          of priority order
    //sum of weight calculated by weight decision table
      is the priority estimation value(Si)
    SumofWeight( );
    PriorityOrder( ); //this method decide priority
                      order by Sumofweight( )'s result
    while ( true ) { //port initialization for
                      connector interaction
    Port p;
    Request r = null;
    Notification n = null;
  }
}
```

Fig. 4. Priority connector interaction using Prism-MW

4 Applied Result and Analysis

In order to compare this proposed technique with the existing technique, we embody the conventional FIFO method that does not consider efficiency characteristic and the proposed method in this paper considering priority in Prism-MW. Measurement results of efficiency characteristics of example components according to these conditions are represented in Table 3.

Table 3. Efficiency characteristic of example EJB component

	EJB Component ID									
	1	2	3	4	5	6	7	8	9	10
Bean request processing time(ms)	809	790	1255	1708	1309	955	763	782	920	860
Memory use rate (%)	7.28	7.20	8.31	8.33	8.25	8.03	6.96	7.02	8.05	7.48

In this test result, Process waiting time of components is shown in Figure 5. In conventional FIFO method, the average waiting time of component processing is 4689.2ms and in this proposed technique, it is 3921.9ms. This result shows that the average waiting time of this proposed technique is shorter.

These results shows that in the aspect of the whole system operation, this proposed technique could carry out more component requests at the initial stage and make the process waiting time of each component shorter. Also, by processing the component with small memory load first, fast processing and quick response could be possible at the initial stage.

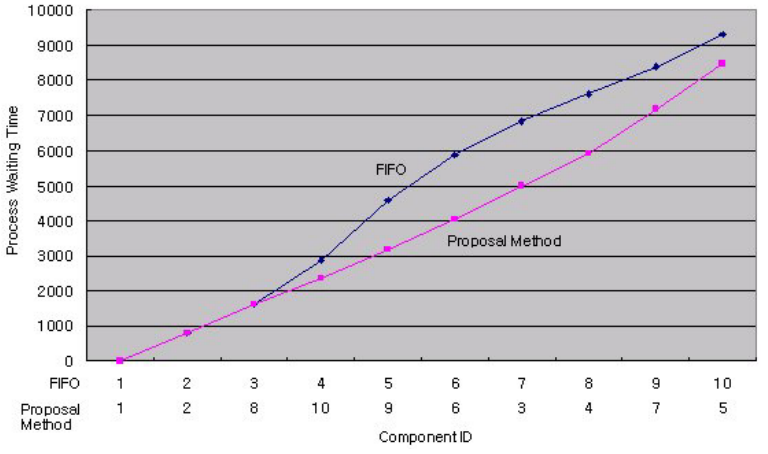


Fig. 5. Process waiting time of components

5 Conclusions

In order to efficiently operate the interaction between components in composition and operation of the architecture-based component, this research suggests the technique that handles the requested processing of component in the connector by the priority. We set up the Prism-MW according to weight decision table on the basis of bean request processing time and memory use rate that have much effect on the process among efficiency characteristics of component. In order to handle component requests, we set the order according to the priority on the basis of component characteristic value.

But this proposed technique only uses bean request processing time and memory use rate among the efficiency characteristics of component. Therefore, as future research, the priority algorithm according to various functional characteristics and efficiency characteristic appropriate for component composition characteristic and the design of the connector that can be operated efficiently in the composition system of a large scale are required.

References

1. C. Szyperski, "Component Software Beyond Object-Oriented Programming", Addison-Wesley / ACM Press. (1998).
2. N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a Taxonomy of Software Connectors. International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, June 4-11. (2000).

3. N. Medvidovic and R. N. Taylor, ”A Classification and Comparison Framework for Software Architecture Description Languages”, IEEE Transactions on Software Engineering, vol. 26, no. 1, January. (2000).
4 Bridget Spitznagel and David Garlan, "A Compositional Formalization of Connector Wrappers", Proceedings of the 2003 International Conference on Software Engineering, (2003).
5 Nenad Medvidovic, Marija Mikic-Rakic, Nikunj Mehta, Sam Malek, "Software Architectural Support for Handheld Computing", Computer, IEEE Computer Society, September, (2003).
6 Nenad Medvidovic, Sam Malek, and Marija Mikic-Rakic. "Software Architectures and Embedded Systems." Proceedings of the Monterey Workshop on Software Engineering for Embedded Systems (SEES 2003), Chicago, IL, September 24-26, (2003).
7 Marija Rakic, Nenad Medvidovic, "Increasing the Confidence in Off-the-Shelf Components: A Software Connector-Based Approach", Technical Report USC-CSE-2000-518, University of California, Irvine, November, (2000).
8 A. Ramdane-Cherif, L. Hazem, N. Levy, “Knowledge Repository Concerning Architectural Styles For Building Component-Based Systems”. CoLogNET’02: Colognet Joint Workshop on Component-Based Software Development and Implementation Technology for Computational Logic Systems. September, (2002).
9. Bakken, David, “Middleware.” Chapter in Encyclopedia of Distributed Computing, Encyclopedia of Distributed Computing, Kluwer Academic Press, (2003).
10. Aniruddha Gokhale. et. al., "Middleware for Communications", Edited by Qusay H. Mahmoud, John Wiley & Sons, Ltd, (2001).
11. Sam Malek, Marija Mikic-Rakic, Nenad Medvidovic, “An Extensible Framework for Autonomic Analysis and Improvement of Distributed Deployment Architectures”, In proceedings of ACM SISGSOFT Workshop on Self-Managed Systems (WOSS 2004), Newport Beach, CA, Oct, (2004).
12. Marija Mikic-Rakic, Sam Malek, Nels Beckman, and Nenad Medvidovic, "Improving Availability of Distributed Event-Based Systems via Run-Time Monitoring and Analysis.", Proceedings of Twin Workshops on Architecting Dependable Systems (WADS 2004), Edinburgh, UK, May 25, 2004 and Florence, Italy, June 30, (2004).

ScudWare: A Context-Aware and Lightweight Middleware for Smart Vehicle Space

Zhaohui Wu, Qing Wu, Jie Sun, Zhigang Gao, Bin Wu, and Mingde Zhao

College of Computer Science, Zhejiang University,
Hangzhou, Zhejiang, China 310027
{wzh,wwwsin,sunjie,gaozhigang,branwu,zmdd48}@zju.edu.cn

Abstract. With computing becomes more and more ubiquitous, it leads to the highly dynamic and complex computing environments. In this new computing circumstance, the need for novel middleware is widely recognized. Especially, applications should be more context-aware and based on the lightweight middleware infrastructure. In this paper, we present ScudWare, an efficient context-aware and adaptive middleware for ubiquitous computing in smart vehicle space. ScudWare is based on smart OSEK OS, ICAR hybrid network and TinyORB. Its main property is context-aware service based on ontology, which can gather, manage, and disseminate context information for applications easily. Using ScudWare ontology, we can easily tackle issues, such as wide variations in information quality. In addition, we bring forward further discussion of the smart vehicle space in ubiquitous computing environments.

1 Introduction[1]

In ubiquitous computing environments, people use smart embedded devices interacting seamlessly with space and devices naturally and transparently. Ubiquitous computing comprises many areas. However, in its core, the active environments, smart devices, and people are essential elements. They adjust themselves continuously to better cooperation and communication. Such user-centered ubiquitous computing environments may require a context-aware and lightweight middleware to communicate, operate resources and provide various services for real-time smart vehicle space [1].

In smart vehicle space, traditional middleware solution is heavyweight, which has not adaptive and reconfigurable mechanisms. In addition, the applications in the vehicle space would require the middleware to execute more reliably and efficiently. At present, many research efforts have focused on designing new middleware architecture for smart space capable of supporting the requirements imposed by ubiquitous computing. Gaia [2] coordinates software entities and heterogeneous networked devices contained in active physical space, exporting services to query and utilize exist-

[1] This research was supported by 863 National High Technology Program under Grant No. 2003AA1Z2080, "Research of Operating System to Support Ubiquitous Computing"

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 266-273, 2005.

ing resources. Active Badge [3] and ParcTab [4] are two projects aim to detect the location of people with a mobile device. However, in our opinion, in ubiquitous computing environments such as smart vehicle space, an ontology-based context-aware and lightweight adaptive solution is needed.

The structure of the paper is as follows. Section 2 presents ScudWare infrastructure, highlighting smart vehicle space, and the middleware architecture is given. Section 3 introduces ontology-based context-aware mechanism. Following this, section 4 gives a case study. Finally, section 5 summarizes the discussion and introduces some areas demanding further investigation.

2 ScudWare Infrastructure

ScudWare consists of smart vehicle space and hierarchy architecture. Once people go into smart vehicle space, they communicate with the smart embedded devices and environments naturally through ScudWare infrastructure. People use services provided by this active space easily. The status of the devices may change spontaneously and dynamically. Following, we give a detailed introduction of above aspects.

2.1 Smart Vehicle Space

Smart Vehicle Space is a workspace, which is embedded in computing devices, information devices and various sensors. This active space makes it convenient for users to access information and service to work alone or cooperate with others. [5] [6]

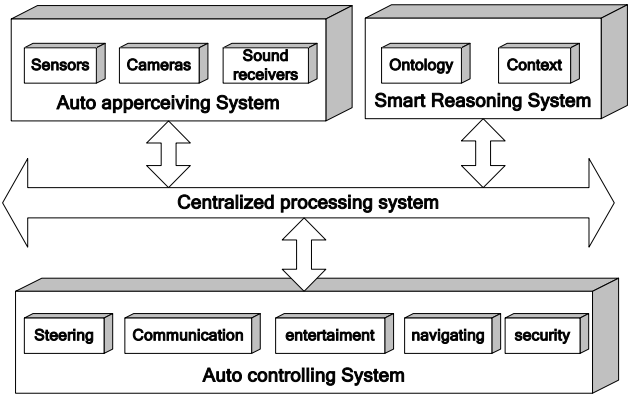


Fig. 1. Smart Vehicle Space

It is composed of (1) auto apperceiving context system, (2) auto controlling system, (3) context repository reasoning system and (4) centralized processing system, as shown in Figure 1. Auto apperceiving context system aims to sense the status of the environment, people and devices in the vehicle, including cameras, sensors, and sound receivers. Auto controlling system comprises steering, communication, navi-

gating, entertainment, security subsystem. Smart reasoning system uses correlative context and ontology-based technology to make decisions. Particularly, the centralized processing system is the kernel for the smart vehicle space, controlling other parts to communicate and cooperate effectively.

2.2 ScudWare Architecture

As Figure 2 shows, ScudWare is made up of four layers. The lowest layer is a real-time operating system named Smart OSEK, which is developed by us. [7] The next layer is ICAR hybrid network, which consists of six parts classified by their different functions as shown in figure 2: Security assurance system, communication system, control system, entertainment system, navigation system and environment system. The third layer is TinyORB kernel. The fourth layer is the core services, including ontology-based context-aware, discovery mechanism and smart reasoning toolkit.

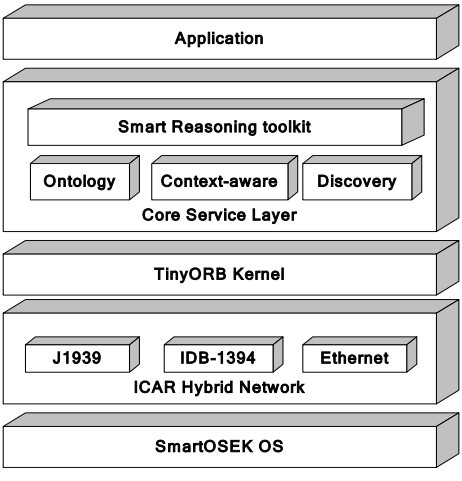


Fig. 2. ScudWare Architecture

Smart vehicle space is an integration network system; we call it ICAR Hybrid Network. It consists of six parts according to the difference of their functions as shown in figure 2: Security assurance system, communication system, control system, entertainment system, navigation system and environment system. Because of their different communication requirements, it is necessary to use J1939, IDB1394 and Ethernet. The information exchanged in control system and environment system is not heavy, so the speed of J1939 is enough to meet their needs. Furthermore, there are some standards defined for in-vehicle control parts; it is favorable for communication designing. Ethernet is used in security assurance system. Because there are video and speed information, the big bandwidth is required. At present, the bandwidth of Ethernet in general has been up to 100 Mbps or 1000 Mbps, which is enough for the transmission of multimedia information. IDB1394 is a popular bus standard for in-vehicle entertainment system. It has the transportation capacity of 400 Mbps, which

can be used to connect parts of entertainment system. The navigation system and communication system are connected to in-vehicle directly. There is an in-vehicle computer, which is a center control unit and can process the information coming from other parts. This computer acts as an interface between human and in-vehicle network system.

2.3 TinyORB Kernel

TinyORB is a light version of real-time ORB (TAO). In addition, TAO is implemented with reusable frameworks from the Adaptive Communication Environment host infrastructure middleware toolkit. We keep the core service such as persistence, real-time scheduling, fault tolerance, and we develop domain-specific middleware service for smart vehicle space, mainly about discovery and ontology-based context reasoning. In TinyORB, ontology-based context-aware object, distributed discovery and peer-matching object are essential issues. We can use unicast and broadcast mechanisms by special policy to inner-object communication, event notification and cooperation. The devices and services can find what they are interested in easily and discover compatible peer services without relying on an infrastructure or centralized approach. TinyORB supports client-server and context-aware P2P computing model.

3 Context-Aware Framework

As the rapid development of embedded technology, it is not a dream for devices to be intelligent. The idea of ubiquitous computing can be realized through multiple kinds of intelligent devices, such as the Media Cup, which can sense the temperature of coffee and notice the person. If daily objects can sense outside environment and become adaptive to the change of environment, it will provide more and better services to human being. With the development of sensor and wireless communication technology, mobile devices are widely used, such as cell phones and PDAs, with the ability to be context-aware, which means, the same device can provide personal and customized services to different users. It can also liberate people from trivial things, since the devices around human can predict his behaviors according to the environment he is in and the activity he is taking on, and react beforehand. Given the ability augmented, it can even make decisions in the emergency quickly for the safety of human beings.

3.1 Overview

According to Anied Dey, context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.[8] The most important and widely used context is location and identity. In addition, time, temperature, activity, the emotion of person, the ob-

jects in the environment are all context. The dynamics can be divided to three classes. Communication dynamics is about bandwidth, error rate, connection setup time, usage costs, security requirements, contention, disconnection rate, round-trip delay. Environmental dynamics comprise of people presence, social situation, and physical conditions. Location dynamics include the entire element associated with concrete location, such as mobile IP allocation, authentication, and social influence when the location of an entity is changed.

Context-aware is the ability to sense and use context. Any application that takes advantage of context is context-aware application. Context-aware computing is the ability of computing devices to detect, interpret, and respond to the change of environment and system.

3.2 Ontology-Based Context Representation

Smart vehicle space is a special environment, in which we can examine the characteristic by the context: The inside of vehicle is a relatively confined and cabined space, the devices inside are limited and fixed, and the space for user to move is restricted. The outside of vehicle is always in a status of rapid movement, but we need not concern all the dynamics in the process. For example, we need not record all the people when driving to work. It is not the target of smart vehicle space. What we need are only those that can influence our system.

The characteristics above determine the definition of context in smart vehicle space. We use the idea of ontology to describe the context information. The concept of ontology comes from philosophy and means theory of existence. The new meaning is provided by computer science region, such as AI. The most accepted definition is "ontology is an explicit specification of a conceptualization [9]". We can understand ontology from two different aspects. Firstly, it is a vocabulary, which uses appropriates terms to describe entities and the relations between entities. Secondly, it is the knowledge base of a specific domain. Ontology is domain-oriented, which describes domain knowledge in a generic way, and provides agreed understanding of the domain.

In our method, we divide the context of smart vehicle space to three parts: vehicle context, environment context, and driver context. Vehicle context concerns the status and attributes of the devices inside the vehicle, such as air-condition, wiper, light, engine, ABS, and the seat. Environment context concerns all the environmental elements, which may influence driving, such as the weather, the road status, the fingerpost, and the signal lamp. Driver's context concerns the status of driver, such as the suitability and ability to drive, physiological parameters such as alcohol density, and pupil diameter. We have used protégé to build context of smart vehicle space and create instances. Protégé is an ontology editor tool, which provides GUI for the user to create and manage ontology architecture. We also use Protégé to export files in OWL format for smart reasoning. Figure 3 is the ontology described by OWL.

```
<owl:FunctionalProperty rdf:ID="NetworkBandwidth">
  <rdfs:domain rdf:resource="#Network"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >the unit is MBPS</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="DeviceState">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >open:begin to play a music according to the filename and volume configured
close:</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Multimedia"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="AbilityOfDriving">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >the ability to drive
may be divided to several levels</rdfs:comment>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Driver"/>
</owl:FunctionalProperty>
```

Fig. 3. Ontology described by OWL

We define context by C=(S, P, O). S denotes the center of context. We use P to describe attributes. O is the value of attribute, or the operator, which is logic operator. We also define the interested and important context scenario by SC= (sc_1, sc_2,…, sc_n) and sc=$\cup c_i$. The scenario integrates the situation of system and the entities involved. We define predication $c_{n+1}=f(c_1, c_2,\ldots, c_n)$ and c_{n+1} denotes the high level context reasoned or the reaction the system should take according to current context scenario. For example :(Driver, Property, Alcohol) $\wedge$ (Alcohol>= 0.04％) $\rightarrow$ (Driver, status, Drunk). It descries if the driver's alcohol level over 0.4, we conclude that he has got drunk. Another, (Environment, Subclass, RoadSurface) $\wedge$ (RoadSurface, Property, Gradient) $\wedge$ (Gradient >= -15%) $\rightarrow$ (Vehicle, Subclass, Brake) $\wedge$ (Brake, Property, BrakeState) $\wedge$ (BrakeState=light). This describes that the car is climbing down a hill when the gradient is bigger than 15 degrees and specifies the brake to be set to a slight work state.

3.3 Context-Based Process Method

In ScudWare, we divide the process into five stages: (1) Context acquisition (2) Context storage (3) Context aggregation (4) Context analysis (5) Context-aware action.

Context acquisition: The stage is to gain the raw data from sensors and transform it to ontology entities or attributes. The data imported are of different type and structure, which must be abstracted so that it can be used by application.

Context storage: The acquired context must be stored to a repository for access when necessary.

Context aggregation: we aggregate correlative context for a specific entity. The target is to trace and record an entity.

Context analysis: we infer the status of an entity or the intention of a user. In ScudWare, we have defined the logic in advance and detected the context of the sys-

tem. If the current context satisfies the precedent of our logic, we think the conclusion of the logic is satisfied, too.

Context-aware action: We specify appropriate action for every context scenario beforehand. In the running time, whenever the system reaches a scenario, the action will be executed automatically.

As a result, this process is not serial and static, since the execution of the action will lead to new changes of system and bring the system to a new context scenario. Besides, there may be multiple process in progress and will interact with each other. Fig 4 shows this process.

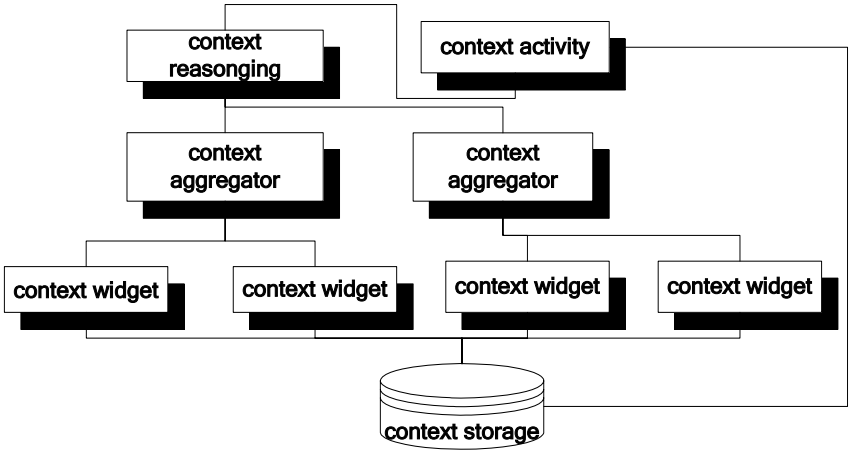


Fig. 4. Context-aware process method

4 A Case Study

To demonstrate the application of ScudWare, we give a case as follows. It is time for Mr. Smith to go to work. He comes in front of his vehicle; puts his palm on the lock authentication machine. His fingerprint is sent to in-vehicle computer. In- vehicle computer receives the data, analyses them, confirms him a legal driver, and then orders to open the door. After the door opens, he gets into the vehicle, sits down, and puts his ID card on the ID machine. At the same time, his weight is gained by sensor and appearance is gained by camera. In-vehicle computer recognizes his identity, so sends welcome command to audio device in entertainment system. A voice “welcome, Mr. Smith” comes from the speaker. At the same time, in-vehicle computer orders entertainment system to play his favorite music and environment system to adjust the air quality according to his favorite. Mr. Smith inputs his destination into in-vehicle computer. The optimal route and some selective routes are given. He selects the optimal one and in-vehicle computer accepts his selection. The apperceiving subsystem detects the vehicle’s work state is good, and reports work state to in-vehicle computer. Then Mr. Smith begins to drive. This is the scenario of our work.

5 Conclusion and Future Work

We have proposed the context-aware and lightweight middleware for smart vehicle space in ubiquitous computing environments. The design of the ScudWare framework is based on Smart OSEK, ICAR hybrid network, TinyORB and core services. Further, we argue that it is crucial to focus on the ontology-based context-aware aspect of cooperation and communication. The service should include providing actively, depending on different context, continuously adjusting adaptively. Next, from a practical point of view, we give a case study of ScudWare.

This is however, a preliminary analysis and further work remains to be done to test ScudWare architecture fully. Our future work includes: (1) improving TinyORB kernel to support secure and reliable application; (2) extending the vehicle ontology-based context presentation; (3) applying more adaptive mechanism to ScudWare.

References

1. Anand Tripathi. Next-Generation Middleware Systems Challenges Designing. Communications of The ACM Vol. 45, No. 6. (2002) 39-42
2. Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan,Roy H. Campbell, Klara Nahrstedt. Gaia: A Middleware Platform for Active Spaces. Mobile Computing and Communications Review, Volume 6, Number 4. (2002) 65-67
3. The Active Badge System.(1992) http://www.uk.research.att.com/ab.html
4. The ParcTab Ubiquitous Computing Experiment.(1992) http://www.ubiq.com/parctab /csl9501/paper. Html
5. Qing Wu, Zhaohui Wu, Bin Wu, Zhou Jiang .Semantic and Adaptive Middleware for Data management in Smart Vehicle Space Proc. the Fifth Advances in Web-Age Information Management, (2004) 107-116
6. Guoqing Yang, Zhaohui Wu, Xiumei Li, Wei Chen. SVE: Embedded Agent Based Smart Vehicle Environment. Intelligent Transport System Conference 2003.(2003) 1745-1749
7. Wu Zhaohui, Wang Lei, Zheng Kougen. A Reliable OS Kernel for Smart Sensors. The 28th Annual International Computer Software and Applications Conference.(2004) 572-577
8. Anind K. Dey. Providing Architectural Support for Building Context-Aware Applications.(2000) http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf
9. Cf. T. R. Gruber. A translation approach to portable ontologies. (1993)Knowledge Acquisition, 5(2):199-220.

Application of Cooperating and Embedded Technology for Network Computer Media Player

Yue Gao[1], Bin Zhang[1], Xichang Zhong[1], Liuying Qu[2]

[1] Hopen Software Engineering Co.,Ltd.
No.4 Southern 4th St, ZhongGuanCun, P.O.Box 2717 Haidian District,
Beijing 100080, P.R. China
gaoyue@mercury.cass.ac.cn
bzhang@hopen.com.cn
xczhong@sec.ac.cn
[2] College of Computer Science and Engineering,
University of Electronic Science and Technology of China
Chengdu, Sichuan 610054, P.R. China
quliuying@163.com

Abstract. A network computer (NC) is a lightweight computer system like a thin client. Although a network computer has many advantages, it is less specialized to play multimedia files through network servers. The conventional mode is that network server not only reads but also decodes multimedia data and network computer just plays frames transmitted from network server. This makes network server overloaded. In this paper, we analyze principles of playing multimedia files between NC and network server, and apply cooperating and embedded technology to resolve the problem. Also, the paper details design and implementation of network computer media player and touches upon the future works.

1 Introduction

1.1 Network Computer (NC)

A network computer (NC) is a lightweight computer system that operates exclusively via a network connection. It only has devices like a monitor, keyboard and mouse, and relies on network servers for storage and software [1]. So the main characteristic of a network computer is that it can strongly depend on power of the network servers and doesn't have its own secondary storage such as a hard disk drive. Thus, we can say that network computer is an example of thin clients[1].

[1] Thin clients are simpler computers or programs which are designed to work with a server, so that the client requires less complexity, local storage, processing, or maintenance.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 274-279, 2005.

A network computer has many prominent advantages. For example, it can reduce the total cost of ownership (TCO[2]) and costs much less than a personal computer. Also, network computers are safer than PCs.

However, a network computer is less specialized to play multimedia files via a network server. Play effects which are displayed on NC terminals are unsatisfactory, even the frames transmitted from the server could not keep consistent. The main reason is relative to principles of playing multimedia files between NC and server. After analyzing, we have found out that the network server is responsible for not only reading multimedia data, but also decoding them. The data decoded is so numerous as to make server overloaded. On account of limitation of network speed and NC's hardware configuration, at the same time, these factors will exactly weaken the qualities of receiving data and playing multimedia files on a network computer. This is the problem to be solved.

Aiming at this problem, cooperating and embedded technology is applied to solve it successfully. We transfer the work of decoding multimedia data to network computer, namely, NC decodes firstly and then plays. Network server just offers multimedia files, while NC completes the work of decoding. In this paper, we will introduce design and implementation of the sofrware of NC media player at length, which has applied cooperating and embedded technology. We will call this software cooperating and embedded media palyer at the following text.

1.2 Remote Desk Protocol (RDP)

In course of implementation of the cooperating and embedded media player, Remote Desktop Protocol (RDP) is used to transmit media player's skin from network server to NC. Then we will introduce principles of RDP briefly.

Remote Desktop Protocol is designed to provide remote display and input capabilities over network connections for Windows based applications running on a server. The performance of the protocol is considered effective by most people for LAN connections. RDP is a multichannel-capable protocol that allows for separate virtual channels for carrying device communication and presentation data from the server. It provides a very extensible base from which to build many more capabilities, supporting up to 64,000 separate channels for data transmission as well as provisions for multipoint transmission [2].

RDP uses its own video driver on the server side to render display output by constructing the rendering information into network packets using RDP protocol and sending them over the network to the client. On the client side, it receives rendering data and interprets them into the corresponding Win32 GDI API calls. On the input path, client mouse and keyboard messages are redirected from the client to the server. On the server side, RDP uses its own virtual keyboard and mouse driver to receive the keyboard and mouse events.

Since requirements of client computer are not exigent, RDP is applied on NC popularly.

---

[2] Abbreviation of Total Cost of Ownership, which is a very popular buzzword representing how much it actually costs to own a PC. The TCO includes: Original cost of the computer and software; Hardware and software upgrades; Maintenance; Technical support; Training [1].

2 Software Architecture

The software of cooperating embedded media player running on a NC consists of three major modules: NC player module, server player module and Self-Defined Transmission Protocol (SDTP). The three modules cooperate with each other and complete the work of playing together.

2.1 NC Player Module

The NC player module interacts with network server via self-defined transmission protocol (SDTP). NC player module receives media files' data and control instructions which are sent from network server, at the same time sends some essential request and response instructions to network server. Then NC player module decodes these data and explains the instructions in order to display on correct location of NC terminal.

Code of NC player module is written on LINUX and embedded in NC.

2.2 Server Player Module

Server player module is mostly responsible for displaying interface of media player which users can operate and control local media files on, and indicating virtual state of playing. At the same time, the module sends user's control instructions and deals with feedback information of users. Server player module dose not play multimedia files indeed, but reads binary data from hard disk of network server and transmits them at suitable speed to network computer. Finally, NC decodes these multimedia data and plays them.

The skin of media player on network server is transmitted to NC using Remote Desktop Protocol (RDP). Area of playing on terminal of NC should cover upon the skin of media player properly.

2.3 Self-Defined Transmission Protocol (SDTP)

For the sake of well communication between NC and network server, we define a transmission protocol----SDTP. The protocol includes operations of communication initialization, sending and receiving of data and instructions, closing connections and so on. By virtue of characteristics of transmission, there are two channels to be defined: one is bidirectional instructions channel based on TCP and this channel is used to transmit instructions and feedback information between NC and network servers; the other is single directional data channel based on UDP and it mainly sends original multimedia data just from network server to NC.

In the abstract, SDTP is on top of Internet protocols, above TCP and UDP. Moreover, the modules of server player and NC player are also on top of SDTP. Detailed relation between three modules and transmission directions of data and instructions are presented at Fig. 1.

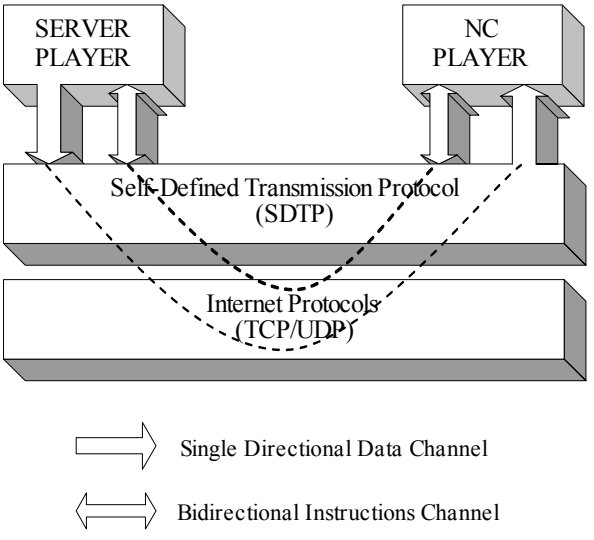


Fig. 1. Software architecture of cooperating embedded media player

Code of transmission protocol is called by network server and NC respectively in mode of library functions. Thereby, SDTP fulfills responsibility of communication commendably.

3 Technical Details

3.1 Communication Procedure

Firstly, NC initializes the library functions of SDTP to monitor some port that is defined beforehand and waits for setting up connection. When users operate the media player's interface on network server, the request of setting up connection is sent to NC simultaneously. After network server has connected with NC, files data and control instructions are transmitted via both of data and instructions channels.

Before sending multimedia files data to NC, network server sends coordinate information of playing area to NC, in order to ensure that playing area can precisely cover upon the skin of media player, which is transmitted from server. Afterwards, NC can decode and play.

3.2 Structure of Loop Storage

Since we use UDP protocol to transmit multimedia data to NC, it is possible to appear that packets are lost and out of sequence during the phases of sending and receiving. As running environment of the software is in a LAN and network is stable in most

cases, the possibility of losing packets is little. However, the problem that packets are out of sequence should not be ignored.

In order to solve the problem, we adopt loop storage structure. According to serial numbers, each packet will be inserted to suitable place in the loop storage structure. Compared with data packets, the size of storage is so large. As a result, those packets will not be ignored even if they are delayed slightly. In this way, it can assure that packets are in sequence in the course of nature.

3.3 Application of Buffering

Buffering plays a critical role in the course of sending and receiving multimedia data. The data packets received from network server are put in a buffer of NC. Meantime, network server sends data packets unceasingly until the percentage of capacitance of NC buffer reaches a specific value (in this software, the value is 90%). Once the percentage value is exceeded, NC sends information of "PAUSE" to network server in time. Until some signal indicates that percentage of NC buffer's capacitance is lower than the value, the network server keeps at sending UDP packets of multimedia data.

4 Test Results and Conclusions

In this paper we have presented the application of cooperating and embedded technology for network computer media player. After a series of tests, it can be certified that the cooperating and embedded media player alleviates burden of server effectively, and improves quality of playing effects on NC terminals obviously. The cooperating and embedded technology is pivotal part in the software.

The hardware and software configurations of server and NC in the experimental environment are presented at Table 1.

Table 1. Hardware and software configuraion of server and NC

Configuration	Server	NC
CPU	Intel Pentium III	ARCA1 166MHz
Memory	256M	64M
Hard Disk	SCSI HD: 37.3GB×1	Nothing
OS	Windows 2000 Server	Orient Software

Aiming at several test targets mainly, we have contrasted the play effects of server decoding multimedia data with that of NC decoding multimedia data. The test results are presented at Table2.

Table 2. Test results of contrasting playing effects

Test targets	Server decode	NC decode
Frames per second	4 fps	15 fps
Color bits	8 bit	16 bit
Resolution	352×288	800×600
Timbre	Slight cacophonous	Pure

5 Future Works

This cooperating and embedded media player in NC presented in the paper is still confined to play multimedia files on network server via LAN. In order to improve performance of network computer media player, we are trying to make this software work on WAN environment and can play multimedia files not only on local server but also on Internet server. Now we are endeavoring to achieve it.

Acknowledgements

This software of cooperating and embedded media player described in this paper is the outcome of work done by NC production division of Hopen Software Engineering Co.,Ltd. We hope that our thoughts about it will be useful to researchers and users of embedded software and system.

References

1. Jupitermedia Corporation. Network Computer. http://ewebnews.webopedia.com/TERM/N/network_computer.html.
2. Microsoft Corporation. Remote Desktop Protocol (RDP) Features and Performance. http://www.microsoft.com/windows2000/techinfo/howitworks/terminal/rdpfandp.asp.
3. Andrew S. Tanenbaum: Computer Networks. 3rd edn. Prentice-Hall International, Inc (1996).
4. Douglas E.Comer, David L. Stevens: Internetworking With TCP/IP Vol3. Publishing House of Electronics Industry (2001).
5. Anthony Jones, Jim Ohlund: Network Programming for Microsoft Windows, Second Edition. Microsoft Press (2002).

QoS Adaptive Algorithms Based on Resources Availability of Mobile Terminals

Yun Li[1] and Lei Luo[2]

[1]School of Computer and Communications Engineering, Southwest Jiaotong University, Chengdu, 610031, China
liy@coretek.com.cn
[2]Computer Science and Engineering College, University of Electronic Science and Technology, Chengdu, 610054, China

Abstract. For mobile terminals, the resources availability always changes continuously because of the mobility and services providing. Based on a resource-aware architecture, the paper proposes an adaptive algorithm using PID and fuzzy control model to adjust the QoS of terminals according to the resources availability. In such architecture, resources can be allocated in stable and agile way. Applications can be aware of resources, and make certain reactions to the change.

1 Introduction

Comparing with the desktop computers, the resources which can be used in mobile terminals are limited because of design requirements of low power consumption, lightweight and small physical size. Although the resources will be enriched with the continuously technical development, desktop computers without the consideration of size, power consumption and weight can be benefited more from the development [1]. Furthermore, terminals are also limited from the network circumstances. The available network connectivity and remote resources change dynamically. In a word, the available resources of mobile terminals change continuously in dynamical and unpredictable way. The terminal should be aware of resources, reacting early when facing the decreased resource availability, and taking advantage during the increasing of the resources.

The paper proposes an adaptive algorithm for mobile terminals based on a resource-aware architecture using PID and fuzzy control model. In such architecture, applications can be aware of resources, reacting early when facing the decreased resource availability, and taking advantage during the increasing of the resources. So the QoS of terminals can be adjusted according to the resources availability.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 280-286, 2005.

2 Resource-Aware Systems

Fig.1 is the architecture with resource-aware capability, which can be used to trace resources and allocate resources to applications according to the availability of resources. *Resource Monitor* monitors terminal resources and provides the information to *Resource Manager* in quantity way. *Resource Manager* allocates resources for application tasks according to some specific strategy to satisfy the situation when multitasks compete to grasp the same resource. According to allocated resources by *Resource Manager*, application can react and adapt to the resource amount at last.

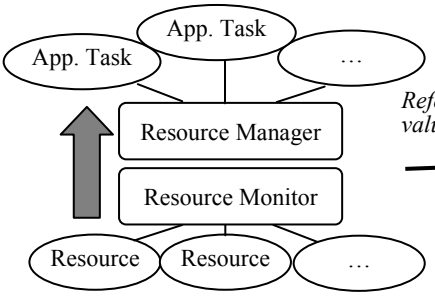


Fig. 1. Architecture of resource-aware systems

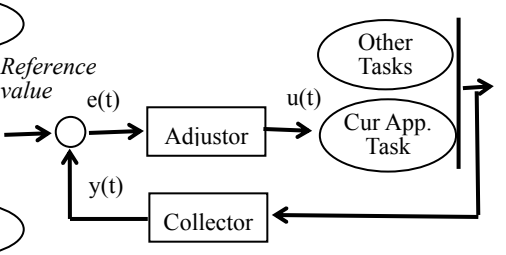


Fig. 2. Resource Manager

The flexibility of Fig.1 is achieved by keeping the *Resource Manager* outside the *Resource Monitor*. This separation decouples application and *Resource Monitor*, which allows *Resource Manager* to update the resources allocation independently without worrying about the application. And an application can make its own action decision on its on-demand resources independent of the existence of *Resource Manager*.

Resource Monitor provides the availability of resources in the form of resource object shown in Fig.3. By this structured form, new resources can be expanded easily and applications can get the resource information according to interested abstract level along the inheritance relation. Further more, the independence among the different parts of systems can be further advanced.

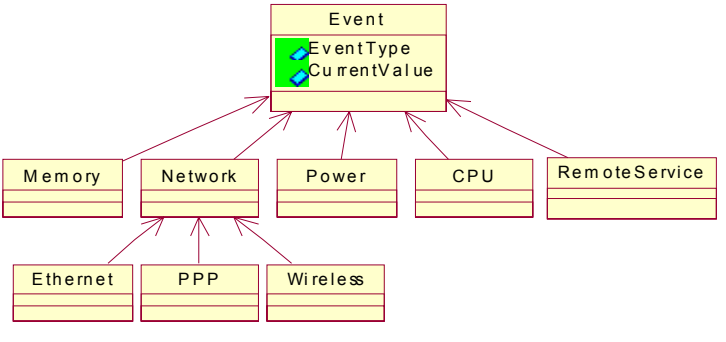


Fig. 3. The Resource Object Hierarchy

The *Resource Manager* can provide fairness and stability in resource allocation by adjusting decision based on control theory. Shown in Fig.2, *Resource Manager* has two components: *Adjustor* and *Collector*. *Adjustor* allocates resources to application tasks dynamically based on resources amount provided by *Resource Monitor* and status of application tasks related to resource consumption collected by *Collector*. If there are competed resources among multitasks, *Adjustor* manages them, otherwise it provides the amount of resources to application directly. In control model of *Adjustor*:

1. Use *r* to represent a specific resource, which can be denoted by <*c*, *t*, *y(t)*, *s(t)*, *A(t)*, *N(t)*>. *c* means the capability of resource, presented by the available largest number of resource. *t* means the period of resource control, and for the reason of simplification, it is a discrete number. *y(t)* means unsatisfied amount of *r* requested by all tasks in the system at the time *t*. *s(t)* is the set of active tasks which compete resource *r* at time *t*. *A(t)* is the task set in which requested amount of resource *r* cannot be satisfied by *Adjustor*. *N(t)* is the task set in which allocated amount of resource *r* by *Adjustor* can satisfy the requested amount of tasks.
2. Use τ to present application task, which can be denoted by <*t*, *r(t)*, *u(t)*>. *t* means time. *u(t)* means the amount of resource allocated to τ in time [*t*，*t*+1] by Adjustor. *r(t)* means the amount of resource requested by τ in time [*t*，*t*+1].

So at time *t*, the unsatisfied amount of resource *r* can be presented by Equation (1) approximately [2], [3]. For application task τ_i, the control model in *Adjustor* can use Equation (2) to compute $u_i(t)$ based on PID control theory[4].

$$\frac{y(t)-y(t-1)}{t-(t-1)} = y(t)-y(t-1) = \sum_{i=0}^{S(t-1)} u_i(t-1) - c \qquad (1)$$

$$u_i(t) = u_i(t-1) + \alpha[w-y(t)] + \beta[-y(t)+2y(t-1)-y(t-2)] \qquad (2)$$

In Equation (2), *w* is a reference and the object of control model in *Adjustor* is to keep the *y(t)* near *w*. α and β are configurable arguments to keep the stability and sensibility of resource allocation, which represents integral and derivative setting in PID separately. α controls the speed of transient response. If α is higher, transient response becomes faster, which leads to a more agile and less stable system. β controls the stability, and if β is higher, the system is more stable, but increases damping. α and β can be specified by a theoretical analysis way, but in practice, estimation and experiment are common way.

3 Multi QoS Levels Based on Resource Availability

Generally, algorithms will produce certain outputs with specified inputs. For instance, in sorting algorithms, for specified data elements, the sorting answer will be unique. In mobile terminals, the concept of traditional algorithms should be extended to consider the shortage and dynamical changing of resources. The computing will be suitable for available resources, providing multi QoS levels for consumers. Such as in multitasking systems, using design method of multi executing paths to satisfy hard real-time requirements, tasks can discard non necessary contents in the situation of

shorting CPU resources so as to guarantee the necessary contents can be executed. For this kind of applications, because of the different resources availability, a specified input may result in a series of possible outputs, which is shown as in Fig.4.

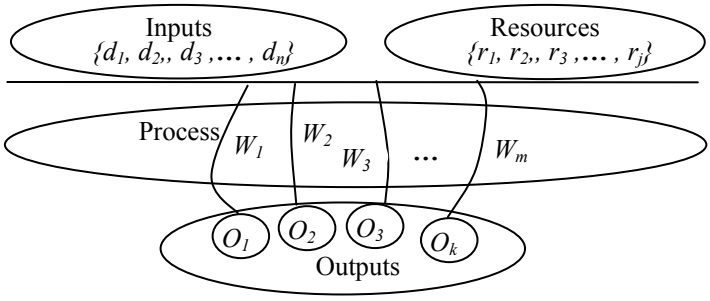


Fig. 4. Multi QoS levels based on resources availability

Models based on fuzzy control are good solutions for these applications. So, applications get resources allocated by *Adjustor* using PID, and consume resources through fuzzy control model.

With fuzzy control model, decisions can be made by linguistic rules in rule base and membership function of linguistic value based on the availability of resources at present. For the control model, the inputs and outputs are all fuzzy sets. So the quantitative resources amount allocated by *Adjustor* should be fuzzified, and the outputs should be defuzzified to generate the decision needed by applications.

Rule base is composed of fuzzy linguistic control rules, which are represented as linguistic values and linguistic variables. To create the rule base, the linguistic rules and the membership functions of linguistic values should be determined. The fuzzy linguistic rules can be described as conditional sentences as following [2]:

$R^{(1)}$: *if* X_1 *is* $A_1^{(1)}$ *and* X_2 *is* $A_2^{(1)}$ *... and* X_n *is* $A_n^{(1)}$ *then* Y *is* $B^{(1)}$

$R^{(2)}$: *if* X_1 *is* $A_1^{(2)}$ *and* X_2 *is* $A_2^{(2)}$ *... and* X_n *is* $A_n^{(2)}$ *then* Y *is* $B^{(2)}$

…

$R^{(m)}$: *if* X_1 *is* $A_1^{(m)}$ *and* X_2 *is* $A_2^{(m)}$ *... and* X_n *is* $A_n^{(m)}$ *then* Y *is* $B^{(m)}$

Here, $R^{(k)}$（$k = 1, \ldots, m$）represents the k^{th} rule; X_1，X_2，…，X_n，Y are linguistic variables; $A_1^{(k)}$，…，$A_n^{(k)}$ and $B^{(k)}$（$k = 1, 2, \ldots, m$）are linguistic values, which are related to fuzzy sets. Each linguistic rule is related to a handling process from fuzzy input to fuzzy output. After conversion, an adaptive decision can be obtained from the fuzzy output.

4 Experimental Analyses

In the experimental environment, there are two application tasks, a browser task τ_1 and an auxiliary task. τ_1 is a periodic task, which displays the web contents periodically, and auxiliary task is not periodic, which simulates the situation of multitasks to compete the same CPU resource, and makes the available CPU resource

of τ_1 changing continuously. The varying network bandwidth is obtained through switching randomly between Ethernet and PPP.

The service quality of τ_1 mainly depends on the following factors: web transfer time and web quality. According to current resource availability, the web quality should be high, and web transfer time should be short at the same time. This object can be achieved by adjusting the contents of browsing web based on the resource availability currently. The resources which should be considered in τ_1 are network bandwidth and allocated CPU resource mainly. In τ_1, the rule base of adaptive decision is:

if NetworkBandwidth is High, then
ContentChoice is *Full*
if NetworkBandwidth is Low and CPU is Idle, then
ContentChoice is GraphCompressed
if NetworkBandwidth is Low and CPU is NotIdle, then
ContentChoice is AlteredTextOfGraph

The membership functions of linguistic functions show in Fig.5[4].

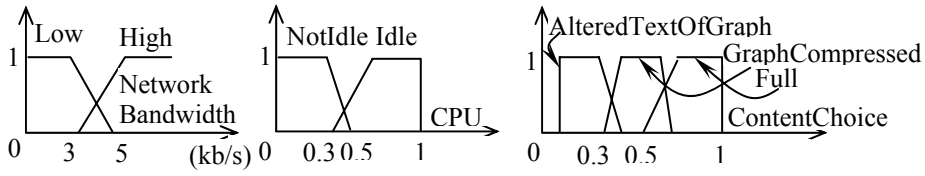


Fig. 5. Membership functions

In experiment, there are three files in Apache Server: *flower.gif*, *flower.jpeg* and *flower.txt*, which represent the same image object in different format. *flower.jpeg* is loss compression of *flower.gif*, and *flower.txt* is the altered text of image. For τ_1, linguistic values *Full*, *GraphCompressed* and *AlteredTextOfGraph* represent the following contents in HTTP requestion, which represent priority to transfer GIF image, JPEG image and altered text respectively:

"Accept: image/gif; q=1.0, image/jpeg; q=0.8, text/plain; q=0.1"
"Accept: image/jpg; q=1.0, image/gif; q=0.8, text/plain; q=0.1"
"Accept: text/plain; q=1.0, image/jpeg; q=0.8, text/gif; q=0.1"

According to adaptive decision, τ_1 request web content according to the above format from Apache Server every 20s. The Fig.6, 7, 8, 9 are the experimental results. Fig.6 means the average data transfer rate. The data are obtained at the HTTP level, representing the average data transfer rate from the requested web to the actual web contents through PPP or Ethernet. Fig.7 means CPU resource allocated to τ_1. At the situation of not using adaptive mechanism, the browser requests web contents by some fixed ways- GIF, JPEG or altered text. The result is that: if the user wants to obtain high quality image, the web transfer time may be too long in the short of bandwidth; if the user wants to shorten the web transfer time, the high quality image cannot be obtained even when the bandwidth is adequate. Fig.8 represents the web transfer time without adaptive considering and Fig.9 represents the web access time with adaptive handling. Using adaptive decision, τ_1 can decide automatically the requested image format according to the current resource availability. So τ_1 can obtain

high quality image if the resource is adequate, and can ensure the short web transfer time even if the resource is poor. By using adaptive mechanism, the browser can obtain stable web browsing performance.

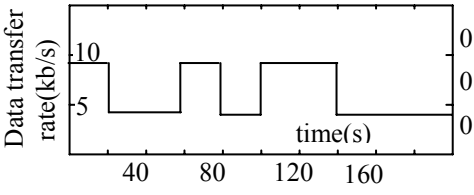


Fig. 6. Average data transfer rate

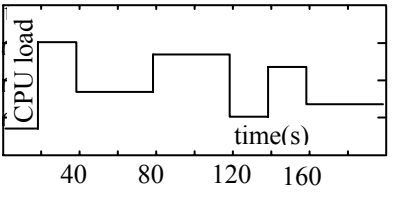


Fig. 7. CPU load

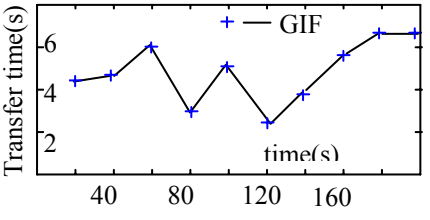


Fig. 8. QoS in general systems

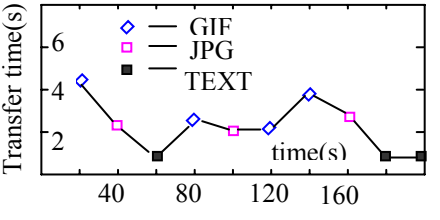


Fig. 9. QoS adjusted by resource availability

5 Conclusions

With the development of network and mobile communication, computing is more flexible and free in resource usage. And pervasive computing will be such a computing mode. In pervasive computing, the terminal is mobile to satisfy the flexibility of service accessing. For mobile terminal, the available resources always change dramatically and unpredictably, which makes the QoS can not be satisfied well.

This paper provides a resource-aware architecture based on PID and fuzzy control model, which allocates competed resources among multitasks through a stable and agile way. And by fuzzy control model, application can make different decisions based on the available resources allocated by resource-aware architecture and provide multi QoS levels. Combined with this resource allocation and resource consumption way, a flexible QoS can be obtained.

References

1. Brian D. Noble. Mobile Data Access. Carnegie Mellon University, 1998. http://www.cs.cmu.edu/afs/cs/ project/coda/Web/docdir /bnoble-thesis.pdf.

2. Baochun Li, Klara Nahrstedt. A Control-Based Middleware Framework for Quality of Service Adaptations. IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms, 17(9)(1999)1632-1650
3. LI Yun, LUO Lei. Resource-Aware Technology for Mobile Terminals. The First International Conference on Embedded Software and System(ICESS2004), Hangzhou, December 2004
4. Li Ren-hou, Intellectual Control Theory and Method, Xidian University Press(1999)

Semi-Videoconference System Using Real-Time Wireless Technologies*

Cheng Jin, Jiajun Bu, Chun Chen, Mingli Song, and Mingyu You

College of Computer Science
Zhejiang University
Hangzhou, P. R. China, 310027
{chengjin, bjj, chenc, brooksong, roseyoumy}@zju.edu.cn

Abstract. We defined a novel system for wireless videoconferencing in this paper. Compared with normal videoconferencing systems, our approach does not need any visual inputs except a neutral image of the user. Our algorithm automatically calculates user expression features on conference server by corresponding voice audio input. These features are transmitted to end users' mobile sets and final expression synthesis can be done there. Since the large visual data is replaced by a small amount of feature data, a great quantity of data bandwidth can be saved, thus improving communication qualities under wireless conditions.

1 Introduction

Teleconference and videoconference are no longer new concepts in modern society. They have been widely used in many areas, such as training, business meeting, medical operations and etc. These applications usually need instant communications between people from difference places.

Generally speaking, a teleconference is defined as a telephone meeting among two or more participants. Sometimes it can also be held while one or several participants sharing a same speaker phone. Without any doubts, this technology is much more complicated than a simple two-way phone connection. Currently, the most common way to hold a teleconference is all participants make phone calls to a pre-arranged server and the server will handle vocal signal dispatching during the meeting. This technology requires low bandwidth of the network because only vocal signals need to be transferred during the conference. However, since this monotype of signal (vocal) greatly limits the information (facial expression, gestures and etc.) carried, this also causes difficulties in communication, especially when there are many participants involved in the meeting, or these participants are not familiar with each other.

* This paper is supported by National High Technology Research by National High Technology Research and Development Program of China (863 Program, No. 2004AA1Z2390) and Key Technologies R&D Program of Zhejiang Province (No. 2005C23047 & No. 2004C11052), and HP Labs.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 287–293, 2005.

A teleconference can be a conference, called a videoconference, if provided with more equipments and special arrangements. In videoconference, participants can see still images or motion videos together with audio signals of each other. By introducing multiple types of signals, participants can "see" the current speakers as if they were sitting in a same room, reducing the possibility of misunderstanding. Since video signals are introduced into the conference, this technology requires relatively high bandwidth to transmit both signals. Despite the high maximum data bandwidth, up to 54 Mbps, that the new IEEE standard 802.11g defines, the maximum operating range of Bluetooth is only 300 feet under normal conditions. [2] It also needs special equipments such as video cameras to capture visual information. Although these equipments are getting cheaper and smaller, these factors still make it sometimes very inconvenient for applications, which may need wireless connections.

In this paper, we try to solve this dilemma by raising the idea of Semi-Videoconferencing System (SVS). The rest of the paper is organized as follows: Section 2 introduces some essential background knowledge and algorithms, which are the fundamental of building the system. The detailed system structure, including the network architecture, is described in Section 3. Finally, section 4 gives a conclusion and raises some interesting topics for further developing.

2 Preparation Knowledge

Section 1 points out that the main problem lies in the balance between bandwidth, or "size of data" in other words, and user satisfaction, which depends greatly on types of signals received.

In order to transmit more types of information, larger bandwidth will be required. For wireless applications, the recommended bandwidth for video is usually 2 times or even higher than that for voice audio, both in GPRS and in UMTS. Table 1 and 2 from [4] illustrate recommended 3GPP and RealMedia Settings for Wireless Networks, respectively. At the same time, the video frame rate is only about 5 to 8 FPS, which is far from satisfactory of human perception.

If the visual signals can be reduced or be replaced by comparatively much smaller ones, the overall throughput can be tripled or even at a higher rate. Song et al [6] attempted to synthesize highly detailed real-time human expressions and facial motions on a given 3-D model by analyzing inputting voice signals. In wireless applications, however, it is neither necessary nor possible to obtain such results of high resolution. For instance, the size of the screen of a PDA is usually 128mm x 96mm (SQCIF, Sub Quarter CIF), which means for a 5-party conference, each side will be shown in a maxim of 64mm48mm area in average. Achieving high-resolution results in such a small area is relatively not cost-effective.

Meanwhile, recent research shows that audio signal may have greater influence on human perception than its visual counterpart. Sometimes, it can even cheat human brains to make visual illusions. [1] It is also the reason why people will not feel too much confused when watching dubbed films, because they tend

to care more about voice, facial expressions and gestures instead of detailed motions such as real movements. This indicates the possibility of using synthetic faces to replace real images while playing voice audio streams simultaneously, if the results of synthesis are good enough to make human eyes/brains to believe they are the "real" facial motions for the corresponding voice signals.

A novel way of expression mapping was introduced by Liu et al [3]. Given a set of frontal images, or a facial expression library L, of one person A and an image of frontal neutral face of another person B. Liu's algorithm can automatically synthesize a same set of facial expressions of B for each expression in L. We apply this idea to expression mapping in video sequence synthesizing. Obviously, it costs too much to perform mapping algorithm for every single frame, and it is also not practical to setup such a large library that can include every possibility of facial expressions. That is why the choice of using a set of predefined key-frames as expression library is preferred. The connection of neighboring key frames is operated by view morphing. [5]

Table 1. Recommended 3GPP Settings for Wireless Networks

Network	Total Bit Rate	Voice Audio	Music Audio	Video	FPS
GPRS	15-25 kbps	AMR-NB 4750-7950 bps	AAC-LC 8 kbps	H.263 or MPEG4 10-20 kbpsSQCIF or QCIF	5-6
UMTS	50 kbps or less	AMR-NB 10200 bps	AAC-LC 12-16 kbps	H.263 or MPEG439.8 kbps (voice)34-38 kbps (music) QCIF	6-15

Table 2. Recommended RealMedia Settings for Wireless Networks

Network	Total Bit Rate	Voice Audio	Music Audio	Video	FPS
GPRS	15-22 kbps	RealAudio® 5 kbps (sipr2) or 6.5 kbps (sipr0)	RealAudio® 6 kbps (cook8) or 8 kbps (cook0)	RealVideo® 8 10-17 kbps SQCIF or QCIF	5-6
UMTS	50 kbps or less	RealAudio® 8.5 kbps (sipr1) 16 kbps (sipr3)	RealAudio® 11 kbps (cook1) 16 kbps (cook2)	RealVideo® 8 34-41.5 kbps (voice) 34-39 kbps (music) QCIF	6-15

3 Semi-Videoconferencing System (SVS)

The processing procedure of our system is divided into two stages: pre-computing stage and real-time state. This division is proved to make the system more efficiently. Before giving the detailed introductions of these stages, we first illustrate the network architecture for better understanding.

3.1 Network Architecture

There are two kinds of servers in the system, one is Synchronization server and the other is the end-server. These two kinds of servers are connected by high-speed cables.

The end-servers are equipped around the place where the conference will be held. This kind of servers acts as base stations in telecommunication, which connects all the users wirelessly or directly. The responsibility of the end-servers is to manage communications with users and Synchronization server, including sending voice data from the user side to Synchronization server, and sending voice/feature data inversely.

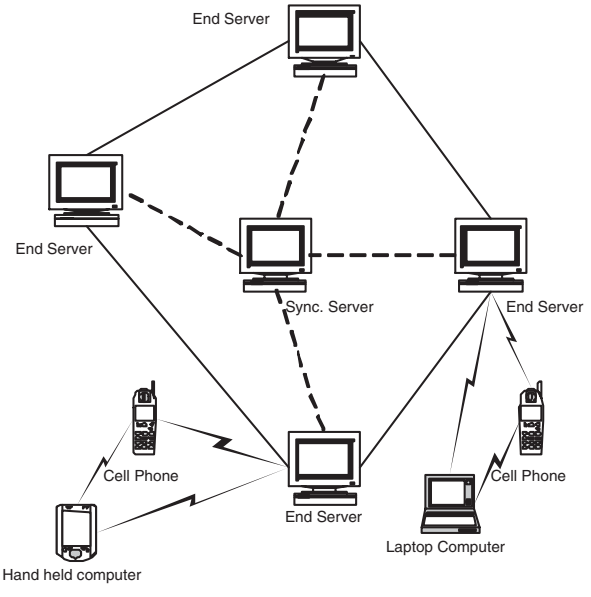


Fig. 1. Network Architecture of SVS

3.2 Pre-computing Stage

The purpose of this stage is to set up a facial expression library (FEL). From the inputting sequential photographs, the system extracts the expression images and

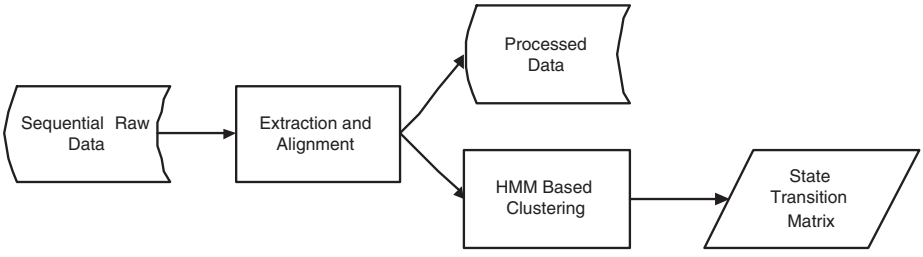


Fig. 2. Pre-computing Stage

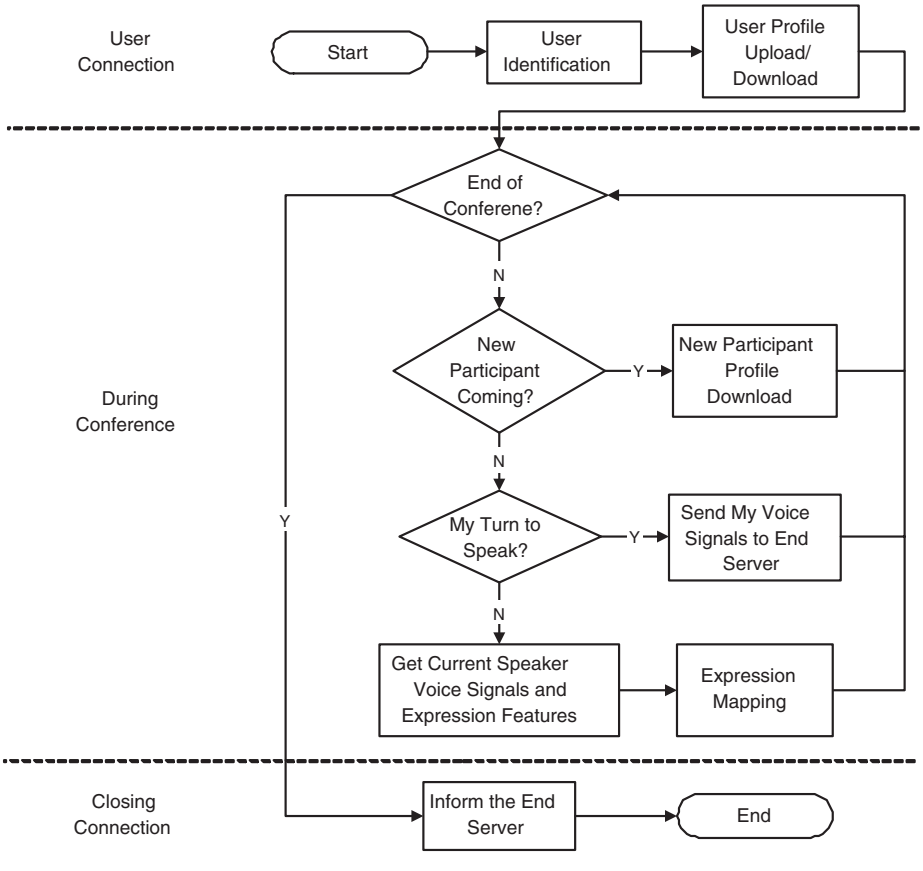


Fig. 3. Real-Time System Overview

aligns them. An HMM based clustering algorithm is applied to divide these images into different categories and then find out State Transition Matrix between them. This procedure makes it possible for the system to predict next expression in real-time. Processed results are stored as FAP/FDP features instead of images in each mobile set, which largely reduces the storage space.

3.3 Real-Time Stage

Figure 3 shows the system's pipeline. After the user is successfully connected, he will be asked to upload his/her profile, such as an image of his/her neutral face. At the same time, the profiles of every other participant will be downloaded to this newly coming user's mobile set.

During the conference when a new participant enters the conference, the system will pass his/her profile to every other participant to ensure everyone gets the most-updated information. When the user is making a speech, his/her voice signal will be transmitted to the end-server. To a certain participant, when other users are speaking, their voice signals together with their synthesized expression features will be passed from the end-server to this user's local mobile set. If there is more than one person speaking at the same time, each voice audio signal is sent from the corresponding end-server to the Synchronization Server. Expression features are calculated respectively for each voice audio there. After it is finished, expression features of different speakers, together with blended voice audio signal, are sent back to each End Server, and then from each End Server to end users that are connected to it. The expression mapping/predicting is performed locally. This is ensured by the computational ability of modern mobile sets, which has been developed to a level that they are capable of handling these complicated jobs. For instance, an HP iPAQ 5555 Pocket PC has a CPU running at 400 MHz with 128 MB SDRAM, which is comparable to a low-end desktop PC.

If the user decides to quit the conference or the conference is finished, the last packet will be sent to the End Server indicating the current user profile can be removed.

4 Conclusions

In this paper, we describe a novel scheme of videoconferencing system for wireless application. The idea of using expression features instead of real video stream is proposed, which greatly reduces the data quantity to be transmitted, and highly improves the desirability of the teleconference.

There are also some further research directions that can be explored in the future. Currently, all data flow directly to the Synchronization Server and the expression feature synthesis job is also done there. This may make the Synchronization Server overloaded. A possible solution is to let End Servers perform expression feature synthesis, and the Synchronization Server plays an organizer role in controlling point-to-point data flow between End Servers.

Another direction is to use Speech-to-Text and Text-to-Speech tools to reduce data size to a much lower level. However, users may want to keep something "real" in communication and there is still much work to do to improve these tools.

References

1. Sound-induced Illusory Flash Perception: Role of Gamma Band Responses. J. Bhattacharya, L. Shams and S. Shimojo, Neuroreport. 2002 Oct 7; 13(14): 1727-30.
2. The New Mainstream Wireless LAN Standard - IEEE 802.11g White Paper, BroadCom, 2003
3. Expressive Expression Mapping with Ratio Images. Z. Liu, Y. Shan and Z. Zhang. SIGGRAPH 2001, Los Angeles, August 12-17. pp. 271-276
4. Encoding Recommendations for Mobile Devices, Version 2.0.2, RealNetworks, 2003
5. View Morphing, S. M. Seitz and C. R. Dyer, Proc. SIGGRAPH 96, 1996, pp. 21-30.
6. Audio-Visual based Emotion Recognition-A New Approach, M. Song, J. Bu, C. Chen and N. Li, Proceedings of IEEE Computer Vision and Pattern Recognition, 2004, pp. 1020- 1025, vol.2, ISSN: 1063-6919

Smart Client Techniques for Online Game on Portable Device

Huacheng Ke, Haixiang Zhang, and Chun Chen

College of Computer Science
Zhejiang University
Hangzhou, P.R.China, 310027
kehuacheng@zju.edu.cn

Abstract. In this paper[1], a smart client model tailored for online game on portable devices is presented. Based on the context of game environment, server-side programming paradigm and smart deployment techniques are proposed to address the problems that emerge in online mobile game development. Network traffic issues of such a model are also discussed in this paper. An illustrative example is given in the last part of the paper.

1 Introduction

Mobile game has evolved from simple single-player, disconnected game to complex multiplayer, online game, as a result of the dramatic marketing growth of portable computing device. Mobile client/server technologies are therefore being applied to mobile game development. In previous research on mobile client/server computing, different models have been proposed. Thin client model like Infopad [6] can solve portability issues but cause high wireless traffic, which is rather expensive for end users. On the other side, full client model like CODA [5], WebExpress [2] reduces wireless traffic at the cost of client development work on heterogeneous platform. For better adaptability, many flexible client models have also been discussed [3]. However, without making use of game contexts, these models are not customized for online games.

In general, online mobile game development has been bottlenecked due to inherent drawbacks of mobile devices and environment:

- *Platform heterogeneousity:* Platforms on consuming portable devices such as J2ME, BREW are highly heterogeneous, preventing application porting and increasing development cost.
- *Low computing power:* Consuming portable devices possesses less powerful CPU and limited storage, which restricts complex game logic and big resources.

[1] This paper is supported by National High Technology Research and Development Program of China (863 Program, No. 2004AA1Z2390) and Key Technologies R&D Program of Zhejiang Province (No. 2005C23047 & No. 2004C11052)

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 294–299, 2005.

- *Wireless connectivity problem:* Wireless connection provides only poor transfer speed, while end users pay much for wireless traffic.
- *Difficulty in deployment:* Online game usually requires frequent update. However, in many mobile game business models, clients are not permitted to modifying downloaded executable binary or will be charged additional fee.

To address these problems in online gaming, we propose a novel model named Anyplay. The primary aspect of the Anyplay model that differs from previous models is the role of portable devices as server managed terminal customized for gaming: Fundamental game elements like graphics, audios are viewed as basic Manipulable Objects in this model. They will be created, destructed and manipulated based on control primitives received by Anyplay client. As a smart client model, the portable client will choose suitable game resources based on its physical features and these resources will be keeping updated based on version check. As a server side programming paradigm, Anyplay provides generic support for playing different games without change in client side.

This paper is organized as follows. After the overview of the whole system, we discuss its detailed design and give some samples and some discussions in packet size. Conclusions follow in the ending part.

2 Overview

As demonstrated in Figure 1, Anyplay system consists of centric game servers and subordinate server-managed terminals. Game developers make game with Anyplay SDK on server side, while the underlying infrastructure and wireless communication are transparent to programmers. On portable devices, platform-dependent terminals are implemented based on common protocols. These terminals respond to server instructions to manipulate basic game elements and send user input to server as well for interaction.

Anyplay model is a server side programming paradigm for online mobile game, in which change to game logic is transparent to clients. As a smart client model, automatic deployment technique keeps clients updated based on version comparison and device capacity. Details of these techniques will be discussed in following sections.

3 Smart Client Techniques for Online Gaming

The main idea of Anyplay model is to make use of game context to address the problems we discussed in the first section. Details of these techniques are explained as follows.

3.1 Smart Deployment

Smart deployment of game resources is the key to reduce wireless network traffic and conquer platform heterogeneousity. Smart deployment consists of two techniques: device adaptation and resource upgrade.

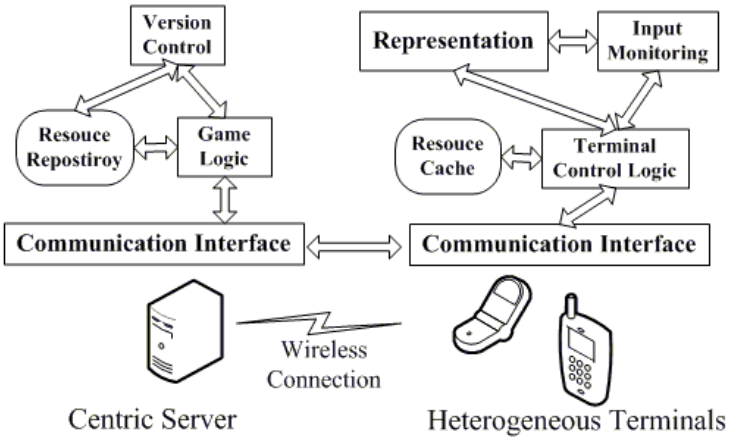


Fig. 1. Anyplay Server and Server Managed Clients

When Anyplay enabled clients connect server for the first time, they retrieve all resources necessary for gaming. We use device adaptation to overcome platform heterogeneousity. In this scheme, the device notifies server with its type ID when trying to establish connection. The server then retrieves the device's physic features and platform information from its database. Therefore, the server is capable of providing different terminals with different resources (image, sound, etc) according to their own features. Physical capacities taken into consideration include hardware features such as computing power, display capacity, and software platform differences as well.

Online gaming requires frequent updates. From the second time of connection on, terminal resources will be upgraded upon connection based on an automatic version check between client and server side. Since Anyplay is a server-side programming paradigm, changes of game logic are also transparent to clients. Therefore this model solves application deployment problem for online games, which requires frequent software updates.

3.2 Resource Cache Management

As the resource cache size in client size is limited, the client can not always download all resources it needs to run a specific game. In these cases, we introduce a priority based scheme to make best use of client cache. Frequently used global resources are marked as higher priority to be kept in cache, while resources that are rarely used will be swapped out when there is no free space in cache.

3.3 Object Oriented Manipulation

In Anyplay, we view gaming process in an object-oriented way: We define a hierarchy of manipulable object classes to describe fundamental elements in gaming.

Each class in such a hierarchy is called Manipulable Object (MO) and categorized into two groups: Entity and Animation. Entity describes substantial objects in gaming context, such as shape, image, audio clip; whereas Animation represents dynamics of Entities: for example, displacement can be applied to graphics entities for game character object movement. We also introduce grouping mechanism in Anyplay model to describe aggregation of objects: EntityGroup for complex objects while AnimationGroup for combined motions. Figure 2 shows a hierarchical diagram for MO in UML [1]

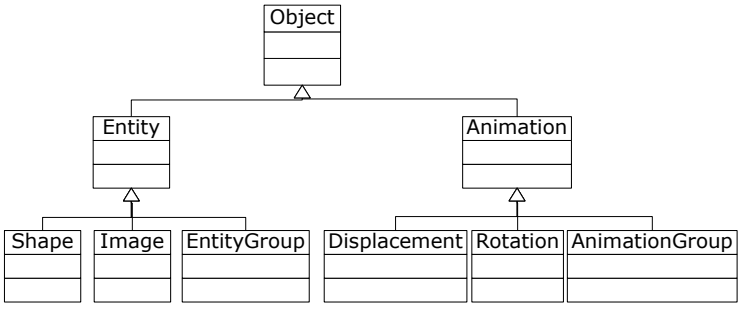


Fig. 2. Manipulable Object Hierarchy (partial) in Anyplay

Anyplay applications adopt a server side programming paradigm. In Anyplay, we define several kinds of basic manipulation primitives:

- Creation: instruct the terminal to create an object, and name it with an identifier
- Destruction: instruct the terminal to destroy an object specified by its identifier
- Animation: animate an entity by connecting it with an animation object
- Binding: bind a single object to an existing group

Formal specification in Z [4] of these primitives is defined as follows:

```
manipulation_primitive = creation_primitive |destruction_primitive
    | animation_primitive | binding_primitive
creation_primitive = "CREATE" + object_encoding + object_id
destruction_primitive = "DESTROY" + object_id
animate_primitive = "ANIMATE" + entity_id + animation_id
binding_primitive = "BIND" + object_id + group_id
object_id = group_id | non_group_id
non_group_id = entity_id |animation_id
```

In Anyplay, All Manipulaple Objects must be created by a Creation Primitive first, then referred by its ID. Creation Primitive actually associates object ID with its encoding. Different Manipulaple Objects are encoded based on their

own characteristics. For example, a circle can be described by its center position, radius, line color and style. However, big resources like bitmap or audio clip are described with an index in pre-downloaded resources package in smart deployment step.

3.4 Scenario-Based Object Creation

By observing Anyplay Primitive specification, it is straightforward that Creation Primitive will consume most bandwidth since it encodes objects rather than simply refer to some IDs. To avoid this, we need to exploit the opportunity within the inherent characteristics of game context.

A scenario-based scheme to avoid unnecessary object creations is proposed. A scenario is defined as "from the perspective of a specific client, a certain period in which a same batch of objects are frequently referred". Therefore objects creation only happens once at scenario start-up and kept in a local cache. Essentially, this scheme takes full use of local storage to reduce network traffic. The choice of scenario usually depends on game context. For example, in an RPG (Role Playing Game), when a game role enters a city, a new scenario should be created and kept sustained until he leaves. In Anyplay model, we encourage game developer to adopt this scenario based scheme by providing a set of APIs for scenario management. Some comparisons on wireless network traffic are given as follow:

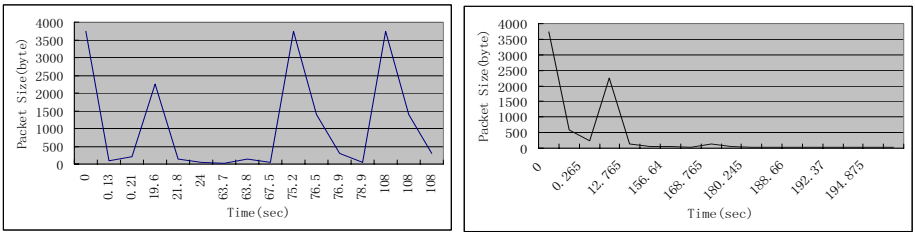


Fig. 3. Network Traffic without (left) and with (right) scenario in a RPG context

4 Example and Packet Size Optimization

Below is an illustrative example of Anyplay enabled multi-player online game in representative heterogenous platform J2ME, BREW and Desktop PC as well.

Meanwhile, in wireless communication, transfer speed is significant influenced by packet size [4]. In Anyplay model, we choose optimal packet size based on experiments on target wireless network. Below is our experimental data on China Unicom's wireless network. Accordingly, our packet size for Unicom is 2 kilobytes.

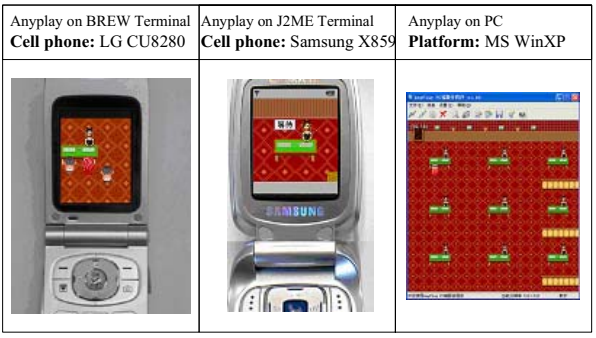


Fig. 4. Anyplay Terminal on Different Devices

Test Environment:
Device: LG CU6260 Cell Phone (with Qualcomm BREW 2.0.1)
Carrier: China Unicom CDMA 1X Network

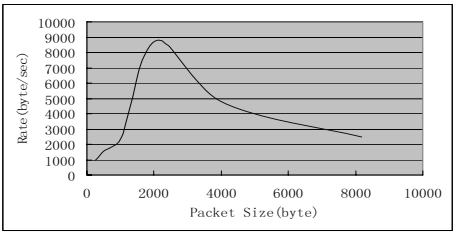


Fig. 5. The Impact of Packet Size on Transfer Speed) in Anyplay

5 Conclusions

In this paper, we present a smart client model for online mobile game - Anyplay. Anyplay addresses inherent disadvantages of online mobile gaming and is optimized for wireless connection. The authors would like to thank Xi Chen and Yi Liu for prototypes construction.

References

1. Martin Fowler, UML Distilled 3rd Edition, Addison Wesley, 2003
2. Housel, et al. WebExpress: A system for optimizing Web browsing in a wireless environment. In Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking
3. Jing, et al. Client-Server Computing in Mobile Environments, ACM Computing Surveys, Vol.31, No.2, June (1999)
4. Modiano, An adaptive algorithm for optimizing the packet size used in wireless ARQ protocols, Wireless Networks 5 (1999) 279-286
5. Satyanrayanan, et al. Coda: A highly available file system for a distributed workstation environment. IEEE Transaction on Computing. 39, 4 (Apr.1990), 447-459.
6. Truman, et al. The InfoPad Multimedia Terminal, IEEE Transactions On Computers, Vol. 47, No. 10, October 1998
7. Woodcock, et al. Using Z, Specification, Refinement and Proof, Prentice Hall, 1996

The Implementation of Mobile IP in Hopen System

Yintang Gu and Xichang Zhong

[1]HOPEN Software Engineering Co., Ltd, 4# South Fourth Street, Zhong Guan Cun, Beijing
P.O.Box 2717, Beijing 100080 P.R. CHINA
gyt75@163.com, xczhong@sec.ac.cn

Abstract. A future wireless network infrastructure is expected to be an all-IP based and supporting mobility between the Base Stations belonging to different types of network as well as belonging to the same network. There're some wireless data infrastructures defined to provide some specific Function Units to manage the IP layer mobility. But most existed infrastructures provide no such Function Units. Here, we consider from the Mobile Station aspect to support the Mobile IP in current wireless network infrastructure.

1 Introduction

The Internet Protocol (IP) represents today's standard in internet networking. Moreover, the IP technology has recently been improved with the introduction of new services such as Voice over IP (VoIP) and best effort/QoS enhancements which represent key features of wireless networks. Thus, the ability to handle data and voice services in an integrated form as well as the possibility of employing the same protocol when accessing the system through a wired or a wireless subnetwork, make extremely attractive solutions based on extensions of the standard IP protocol for mobility management in third generation cellular systems [1-3].

Mobile IP is the current standard for supporting macro-mobility in IP networks [4]. With Mobile IP, the mobile wireless computers can be attached to the Internet and remain attached to the Internet even when they move from place to place, establishing new links and moving away from previously established links.

Such a system must be able to manage the handoff between cells within the same access network as well as between different access networks. Currently, there are some wireless data network architectures, such as GPRS and CDMA being defined, which support mico-mobility with which the WS (Wireless Station) can access the Internet using one permanent IP address while moving within the same IFU (Interworking Function Unit, such as SGSN in GPRS[5] or IWF in CDMA) area and without necessity of re-registering to HA. With implementation of VLR, HLR and adding tunnel function to IFU, these architectures can also provide macro-mobility management, with which MS can communicate with its peer node without changing its IP address when moving between different IFUs.

It's sufficient for MS to only have a sense of micro-mobility provided by every network being accessed through implemented VLR, HLR and tunnel function. Unfortunately, most existing wireless network structures haven't provided these

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 300-305, 2005.

service units yet. So, we consider the mobility support from the wireless MS aspect in this article. We discuss the functions needed by a wireless MS for supporting macro-mobility and micro-mobility, the message sequence and data flow between the MS and the IFU, MS and HA (Home Agent), and the advantages of our implementation.

2 Wireless Data Network Architectures

Here we will overview some wireless data network architectures currently being defined.

General Packet Radio Service (GPRS) is being defined by the European Telecommunications Standards Institute (ETSI) to provide packet data service using Global System for Mobile Communications (GSM) cellular networks. A high-level diagram of a GPRS network is shown in Fig. 1. GPRS uses a combination of link-layer and newly defined higher-layer technique for mobility management.

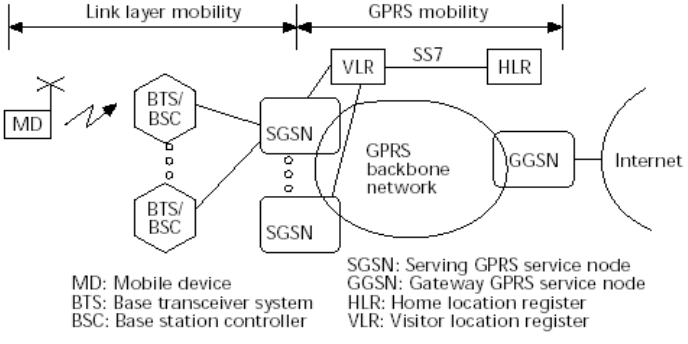


Fig. 1. GPRS Network Architecture

On the air interface, GPRS supports registration, authentication, paging, and handoff (called cell reselection), as well as procedures for channel access to transmit data packets. GPRS allows the mobile host to operate in two distinct states: an active state where the network knows the location of the mobile host's current base station, and a standby state where the network knows only the approximate location of the user, such as a set of base stations, called the paging area, in which the mobile host resides. One of the motivations for defining the standby state is to reduce the host's battery power consumption by allowing the mobile host to only notify the network when it moves out of the paging area. If data packets for a mobile host in standby state arrive at the wireless access network, the serving GPRS service node (SGSN) pages the mobile host in its paging area to determine the mobile host's current base station before delivering the data packets.

In the backbone network, GPRS defines a new tunneling protocol built on top of an IP network, called the Generic Tunneling Protocol (GTP), to handle device mobility, and support registration and authentication procedures. Data packets flowing through the tunnel are encapsulated with an outer GTP/UDP/IP header. This adds 48 bytes of header overhead to each data packet, which is substantial for voice-over-IP

applications that transmit data packets with a small payload. GPRS also defines a QoS profile for each user with attributes for precedence, delay, reliability, as well as peak and mean throughput classes. However, the drawback of defining GPRS-specific QoS support mechanisms is that advances in IP QoS support, such as integrated [3] and differentiated [4] services, may not be directly applicable.

In Fig. 1, the air interface protocols from the mobile device are terminated at the base terminal station and base station controllers (BTS/BSCs) (shown as a single box for simplicity). The GTP tunnels extend between the two GPRS gateway routers: the SGSN terminates one end of the tunnel and directs packets to the proper BTS/BSC using link layer protocols; the gateway GSN (GGSN) terminates the other end of the GPRS tunnel and is a gateway to the Internet. As a device moves between SGSNs, new GTP tunnels are established to manage mobility. As a device moves between BTSs/BSCs on a single SGSN, handoffs are handled at the link layer.1 GPRS reuses the same infrastructure deployed for GSM in order to support authentication, registration, and roaming. In particular, each SGSN is connected to a visitor location register (VLR), which holds a temporary database of the users currently attached to it. A permanent database of registered users is kept in the home location register (HLR), together with a pointer to their current VLRs. Whenever a new user has to be authenticated, the VLR contacts the user's HLR. The HLR replies to the VLR with authentication information, which is composed of a set of random challenges and their corresponding responses, obtained with the use of a secret key that the HLR shares with the user. By sending the challenges to the user and comparing its responses with those obtained from the HLR, the VLR performs user authentication. Similarly, for ciphering between the SGSN and the user, the HLR can send to the VLR encryption keys, obtained from the same secret key known only to the user and HLR.

The CDMA network architecture is similar to the GPRS Architecture. One important difference between GPRS and CDMA networks is that in CDMA networks a mobile device may communicate with more than one base station during a soft handoff, thereby transmitting duplicate data frames and increasing the probability of the correct reception of user data. CDMA networks use HLR/VLR mechanisms similar to GPRS2 for supporting user authentication, registration, and roaming. In addition, CDMA networks use authentication procedures defined for Mobile IP. And data packets between the home agent and the foreign agent are encapsulated using an IP-in- IP tunnel.

3 Implementation of Mobile IP in Hopen System

All the maro-mobility is managed by VLR, HLR and IFU in GPRS and CDMA network introduced above. So it's essential for every access network providing VLR, HLR and IFU to support mobile IP. If the current access network does not provide these Function Units, but just provide the service to access Internet, i.e. MS can obtain an IP address from the current network and communicate with the Internet using the obtained IP, thus the MS cannot enjoy the mobile service. Then, what can we do if we want to enjoy the mobile service in an access network without these Function Units? The solution below may be an answer.

3.1 Assumptions

Before expatiation on the details of the implementation, some assumptions are given as follows.

- The current access network can access Internet.
- The current access network provides some mechanism for MS to obtain a temp IP address.
- HA is provided in MS's home network.
- There's a mechanism between MS and FA with which the MS can be informed if FA is provided in current access network.

3.2 Hopen Protocol Stack

Hopen is an embedded real time OS. And it has been used in some mobile devices such as PDA and mobile phone. Hopen protocol stack support a whole TCP/IP protocol family. The structure of the Hopen protocol stack is showed in fig 2.

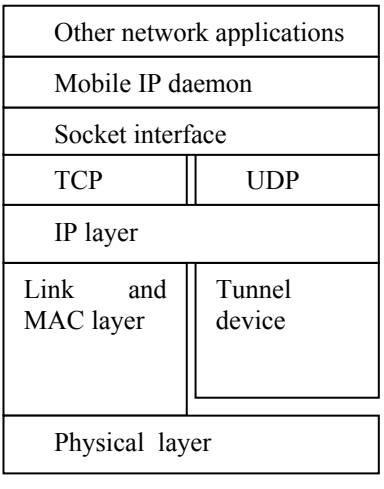


Fig. 2. Hopen Protocol Stack

The physical layer is the MS's interface to BTS. The link and mac layer are network-specific parts, which includes MAC, RLC, LLC and SNDCP for GPRS. The tunnel device is a virtual link layer device supporting mobile IP tunnel. The mobile IP daemon is responsible for registration process and managing the tunnel devices.

3.3 Message Sequence

The message sequence of the MS is depicted as follows.

MS obtains a permanent IP address from its Home Network as its home address. While the MS moves from one BTS to another within its home network, the link layer

protocol manages the mobility, and the layers above IP has no acknowledgement of the mobility.

When MS accesses a foreign network supporting mobile IP, the mobile IP service is obtained at the process of registration to the current BTS as GPRS protocol described. So layers above IP in MS don't know this mobility either. In this case, MS just need to deal with the micro-mobility though the current access network is a foreign network. The macro-mobility is processed by the SGSN, VLR and HLR in GPRS infrastructure or by IWF, VLR and HLR in CDMA infrastructure.

When MS accesses a foreign network which doesn't supply IFU and only supplies the gateway through which the MS could access the Internet resource, the link layer notifies the upper layers that there's no Mobile IP service available in current access network. At the same time, the temp IP address obtained from the current network is informed to mobile IP daemon. The mobile IP daemon uses the temp IP address as the Co-allocated Care-of Address to register to its HA. The IP tunnel is finally created after MS receives the register reply message from its HA.

3.4 Routing and Data Flow

When the MS accesses the Internet from a foreign network not supporting mobile IP, the routing and data flow are described as follows: the packets from the MS to its CN (Correspondent Node) are directly forwarded using the current routing policy. The packets from the CN is first forwarded to the MS's home network and captured by MS's HA. HA resembles the origin packet into tunnel which targets to the Co-allocated Care-of Address of the MS. The tunnel packet is untunneled in the MS's tunnel device and finally the origin packet is sent to the upper protocol of the MS.

3.5 Advantages of the Implementation

There are two advantages of our implementation comparing to the resolutions just supporting the micro-mobility.

- The MS can use mobile IP service in the network not providing FA.
- In current resolutions, every MS needs two global routable IP addresses, HA address and CCA. This makes the lack problem of the number of addresses in IPv4 more serious. Through adding the UDP tunnel module in the MS and HA, our solution can support the network environment described in RFC3519 in which MS's CCA could be private address. Using private address as CCA could alleviate the growth of IP address.

4 Conclusion

Considering the handoff efficiency and the scalability, the new infrastructures for wireless network treat the MS mobility as two parts, micro-mobility and macro-mobility. The MS only needs to deal with the micro-mobility while all the macro-mobility management is performed by some specific Function Units. This prohibits

the MS using Mobile IP in the network not providing these Function Units. Our MS implementation can work in this case without collapsing any performance of these network infrastructures.

References

1. Girish Patel, Steven Dennett: The 3GPP and 3GPP2 Movements Towards an All IP Mobile network. IEEE Personal Communications Interactive (August 2000).
2. Motorola: Wireless Internet Network Communications Architecture. The International Engineering Consortium, http://www.iec.org.
3. Nokia: IP-Radio Access Network . Nokia's vision for an all IP based architecture for Radio Access Networks, http://www.nokia.com.
4. C. E. Perkins: IP Mobility Support. RFC 2002 (Oct. 1996).
5. Digital Cellular Telecommunication System, General Packet Radio Service, Service Description — Stage 2, GSM 03.60 v. 6.0 ETSI (1998).

A New CGI Queueing Model Designed in Embedded Web Server

Xi-huang Zhang and Wen-bo Xu

School of Information Engineering, Southern Yangtze University
214122 Wuxi, Jiangsu, P.R.China
{zxhsytu, xwb}@sytu.edu.cn

Abstract. The embedded web servers play very important part in embedded system. Controlling the timing performance of each individual request, such as CGI (Common Gateway Interface), between client and server is a challenging problem. CGI requests/responses not only are the useable access to interaction with the web server in embedded device, but also are executed in web server in real-timing performance of a network close to the service level specification. To ensure the CGI response within a specified time to meet the needs of real-time demand, the principal of the CGI queueing model in embedded web server is studied, and the approach to the problem of meeting relative delay guarantees in web servers is extended. Furthermore, the process of CGI request executed in server is divided into several subroutines, which is useful to reduce the CGI request response time. This web server has been successfully implemented in embedded platform for a real-time controlling system and the tested CGI response performance shows that the new CGI Queueing Model is efficient.

1 Introduction

In a network based embedded systems, embedded web servers have become an integral part of our information services infrastructure. End-users can execute web-enabled applications from any location by browsers, which has been the standard solution of remote control system. Typically there is a web server in device and a browser as user interface for a variety of applications.

To reduce the cost of embedded system, we usually build a MCU based web server in many embedded appliances [1, 3, 4]. Due to the limited processing ability, CGI (Common Gateway Interface) is used as the basic communication mechanism between servers and client terminals. Remote devices can be definitely monitored with embedded web servers supporting CGI functionality [6]. To ensure the CGI response within a specified time in the need of real-time demand, we should improve the normal execution of CGI request. The Queueing theory and feedback method are widely studied to guarantee controllable delay of the CGI response [2].

In this paper, we first study the principal of the CGI queueing model and generalize the approach to the problem of meeting relative delay guarantees in web servers. Secondly, we suggest a new subroutine model to improve the real-time performance. At last, in embedded environments, we offer a novel approach to implement the CGI.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 306-311, 2005.

2 The Principle of the Queueing Model

Queueing theory provides the predictive framework by which expected delays can be inferred directly from input load [2, 5]. The queuing theory is also used to compute the service rate and necessary resources to achieve a specified average delay when the currently observed average request arrival rate is given. Server resources are then allocated to achieve the computed service rate. A feedback control loop compares the actual delay achieved to the desired average, and therefore adjusts the resource allocation accordingly in an incremental manner to ensure that the desired delay is maintained and completed before deadline. The feedback and queuing components operate concurrently in a complementary manner.

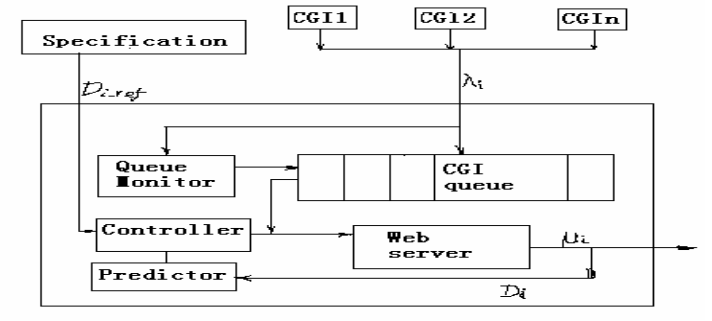


Fig. 1. The queueing model structure in embedded web server

Let the CGI request process (shown at the top of Figure 1) have arrival rate λ_i, which may change abruptly, with a deadline. The queueing model should obtain the arrival rate and deadline online. If the average delay of a queue is less than user specification and the CGI request is completed before the deadline, the web server gets the agreement between the request and the response. If the average delay of a queue is larger than user specification and the CGI request is completed after the deadline, the web server should be corrected by the controller via service rate adjustment and scheduler to get the agreement between the request and the response. Before completing the above adjustment, the mean delay D_j of the request must be computed and compared with the expected delay $D_{i\text{-}ref}$. The average arrival rate λ_i and the server rate μ_i also need to be calculated. For allocation, let $MAXn_i$ denote the maximum number of server processes, that is, the server's max capacity. And let n_i represent the number of server processes stayed in queue. By the feedback mechanism, the model described above is shown in Figure 1.

We have added a group of new modules implementing the queueing model described above. The modules to manage the CGI task queue come from client devices are Queue Monitor, Predictor, Queue Controller and Scheduler. According to the resulting real-time, the queue scheduler determines a certain number of CGI request connections from this client. The queue scheduler runs the highest-priority CGI request selected from CGI request queue. Every CGI request from client has its descriptor to point the CGI request priority and deadline, which are used to schedule

the queue. Queue Monitor is executed to do basic operations on CGI queue such as inserting, deleting and sorting. It is an essential part of the model.

A CGI request would be changed into a server process queued in CGI queue executed in web server. At the mean time, two important modules, Queuing Model Predictor and the Feedback Controller, are necessarily employed to decide the CGI request-response time for the client to be controlled.

2.1 Queuing Model Predictor

The web server should deal with the large number of requests from clients and response these requests in time, which is dominated by the ability of web server. Before the control policy and resources allocation are detailed, Queuing Model Predictor should compute three values.

The first value is the max number of CGI request, which the web server is able to cope with. To estimate the max number of CGI request, we should know the max capacity of the web server C. Assume there are N instances of CGI request, *(*CGI_i *i=1,2,3…N)*. Every CGI request statically needs the recourse c_i or c_i is allocated to CGI request. At a moment, there are n_i instances in web server. It is obvious that the max number of instances for a CGI request $MAXn_i$ is restricted by following formula:

$$\sum MAXn_i \cdot c_i < C \tag{1}$$

Because of the variety of n_i, the max number of the CGI request instances is changeable. By the above formula, the Queuing Model Predictor can compute the max number of the CGI request instances.

The second value is request delay D_i that is used to select an emergent task from queue to schedule according to the deadline. If the web server system know every instances executing time for CGI_i, it is easy to estimate the request delay D_i by the average or max of all the time of CGI_i instants.

Obviously the serve rate μ is the last value to compute. Assume n latest instances of CGI_i are finished from f_1 to f_2. The serve rate μ is described as following:

$$\mu = \frac{n}{f_2 - f_1} \tag{2}$$

Let c be the total capacity of the server and a resource reserve of size $c_i \leq C$ be dynamically allocated to CGI_i request. The corresponding service rate μ_i will be $c_i \mu$, where μ_i is the service rate per resource unit. The long-term average queueing time for the clients CGI request is

$$W = \frac{\lambda}{c_i \mu (c_i \mu - \lambda)} \tag{3}$$

To meet the relative delay guarantee specified by above equation, suggested by [2], the value of c_i is got to satisfy the desired value for the resource allocation.

2.2 Controller and Feedback

The general PID controller known for its robustness and predominant use in industry can be implemented to control adjustment of server process quota because of its elimination of residual errors [6]. Although we can use the general PID controller to adjust the web server resource by some parameters, it should be calculated in web server that slows down the response. Here we present a simple control algorithm called ping-pong method. The Controller computes the errors as following:

- $e_D=D_i - D_{i\text{-}ref}$, $D_{i\text{-}ref}$, means the desired delay,
- $e_M=MAX_{ni} - MAX_{ni\text{-}ref}$, $MAX_{ni\text{-}ref}$ means the MAX_{ni} should be,
- $e=\lambda-\mu$

Based on all above errors, the control policies are described in Table 1.

Table 1. The feedback control policy in the queueing model

If	Control policy
$e_D<0$	Priority of CGI request decreased 1 by feedback Controller
$e_D>0$	The priority of CGI request increased 1 by feedback Controller
$e_M<0$	The max number of CGI request decrease 1 by feedback Controller
$e_M>0$	The max number of CGI request increase 1 by feedback Controller
$e<0$	Adjustment of server process quota by decrease 1
$e>0$	Adjustment of server process quota by increase 1

3 The Subroutine Model

The queuing model can ensure the CGI request being completed before the deadline by the currently observed average request arrival rate. But it is possible for controller to schedule smaller unit in web server because it is well known that a CGI request (a task) could be divided into some subroutines. The subroutines belong to either one of the two categories: non real-time and real-time as we have mentioned. That is to say that a CGI request task could be executed through executing several non real-time and real-time subroutines. In web queueing model system, the real-time subroutine is queued and scheduled with preemption algorithms by priority in the embedded system. We can simply imply that every subroutine could be executed in parallel method, which considerably reduces the executing time of a task because of the less waiting time of I/O operation.

To implement the subroutine in queueing model, Communication and Synchronization are implemented. The communication between subroutine and the environment requires some amount of buffer. When an input is read at the beginning of a subroutine, the value is stored in a buffer and then read by other real-time subroutine or non real-time subroutine. The read operation is non-blocking and non-consuming, i.e., a value will always be present in the buffer and the same value can be read for several times. Similarly, another real-time primitive is used to write a new output value. The value is stored and is written to the output at the end of the relevant

subroutine. The write operation is non-blocking and any old value in the buffer will be overwritten.

Shared variables are used to handle the communications between tasks or subroutines. If an input is associated with a shared variable, the value of the variable is copied to the input buffer at the front of the relevant segment. Similarly, if an output is associated with a shared variable, the value in the output buffer is copied to the shared variable at the end of the relevant segment. The use of buffers and non-blocking read and write operations is allowed with different periods to communicate. The periods of two communicating tasks need not be harmonic, even if it makes most sense in typical applications. If two tasks or subroutines should write to the same output or shared variable at the same time, the write lock is used via PV operation.

4 Implementations and Experimentations

We have inserted a module, which implements the above model to control the CGI client request rate and ensure the CGI being completed before the deadline in embedded web server, as the mechanical center of the embedded web system. This design allows us to control the request queue without basic web server modifications. When the web server boots up, the module creates a process to maintain an HTTP connection request. After a CGI request arrives, the CGI request is put into a queue to wait for an available process scheduled by system kernel. Some scheduler algorithms such as persistent connections are implemented. A single queue is used because of simplification. The simple scheduler as a real-time web server kernel has run successfully for the web server on many platforms even in MCU.

In order to evaluate the performance of the queueing model based on the principal discussed above, we compared its performance with that of the G/G/1 queueing model with PID. And the performance of the subroutine in queueing model is also evaluated.

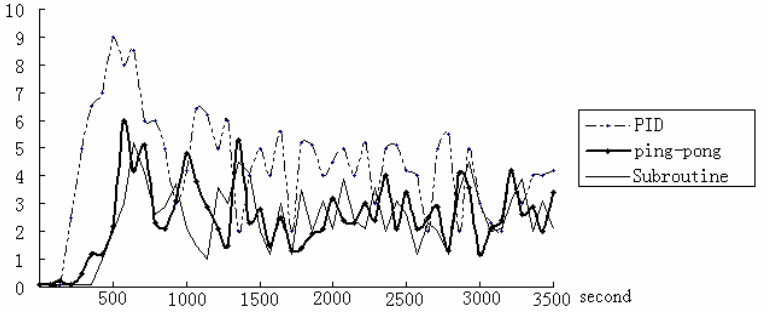


Fig. 2. The average response delays

The Web CGI server scheduler listens to all port and accepts every incoming CGI request. The scheduler maintains an FIFO CGI request queue. In experiments, the desired connection delay for the system is $D_{ref}=4$. The average delays measured in every sampling period are shown in Figure 2.

In the first experiment, we explore the effect of normal PID feedback control for resource allocation. In the second experiment, we explore the benefits of queueing modal with ping-pong feedback. And the subroutine in queueing model is done in the third experiment. The evaluation demonstrates that the advantages of the queueing model with ping-pong method. The combined model is better than the other algorithms such as PID and ping-pong if a task could be divided into non real-time and the real-time subroutines. The average delays are 4.52, 3.21 and 3.15, respectively.

5 Conclusions

It is most significant to design the embedded web servers with low-cost, high-usability properties, which can meet the real-time demands. Due to limited resources available in embedded systems, the system might drop packets of client request or lose control loop and miss the real-time application requirements. These limited resources are largely committed to critical missions and real-time applications in many cases.

In this paper, the CGI queueing model designed in embedded web server has been built to achieve a specified average delay given by the currently observed average request arrival rate. And then to avoid complex PID feedback control, the reduced control policies called ping-pong method was addressed thoroughly. The subroutine model mentioned above can make better performance during feedback control. The simple scheduler with ping-pong feedback control and subroutine model is the main feature of the designed queueing model and has implemented in an embedded web server. This model will speed up the progress to make low-end control devices become interconnected, which is denoted by the experimental results.

References

1. Terry H.Ess., T.H.E.Solution, LLC,Greer,SC.: Accessing Devices Using a Web Service. Proceedings IEEE Southeast Con 2002
2. Ying Lu., Tarek Abdelzaher.: Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers. Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'2003)
3. Ian Agranat. Embedded Web servers in Network Devices. Communication Systems Design, March 1998
4. T. F. Abdelzaher, C. Lu.:Modeling and Performance Control of Internet Servers. Invited Paper, 39th IEEE Conference on Decision and Control, Sydney, Australia, December 2000.
5. Lui Sha, Xue Liu.: Queueing Model Based Network Server Performance Control. Proceedings of the 23rd IEEE REAL-TIME SYSTEMS SYMPOSIUM (RTSS'02)(2002)
6. Xihuang Zhang, Zhilei Cai.: Properties and Implementation of the Embedded CGI. Mini-micro Systems, 2003 Vol.24 No.11 (2046-2048)

A New Embedded Wireless Microcensor Network Based on Bluetooth Scatternet and PMCN

Kangqu Zhou[1] and Wenge Yu[2]

[1] Department of Industrial Engineering, Chongqing Institute of Technology, Chongqing 400050, China
zhoukangqu@126.com
[2] Basic of Logistical Engineering University, Chongqing 400016, China
yuwenge@126.com

Abstract. With the rapid development of wireless network technology and PMCN technology, people are strongly interested in wireless communication and network communication. A new architecture of EWMN is established based on Bluetooth, PMCN data communication, and embedded Linux, etc. The functions and applications of the new EWMN are discussed and studied in detail. The network structure of the EWMN can be divided into three layers: Bluetooth scatternet layer, PMCN layer, and Internet layer. The advantages of EWMN include: long-range application and flexible system configuration, lower equipment cost and communication expenses, and easy to control and manage. The basic key elements of EWMN are analyzed in detail.

1 Introduction

Under the background of global informatization, the profound changes are taking place in the sensor, which exists in the front end of information stream. Being-wireless, networking, synthesizing, and micromation for sensor technologies have become inevitably four development trends. The availability of low-power microsensors, embedded processors, actuators, and radios is enabling the application of wireless network sensing to a wide range of applications, including environmental monitoring, imaging processing, etc. In the future, wireless microsensor networks based on smart environment will play a key role in sensing, collecting, and disseminating information of environmental phenomena. Consequently, wireless microsensor networks will undoubtedly provide new monitoring and controlling capabilities for many applications [1].

With the development of short-range radio frequency network technology and PMCN (public mobile communication network) technology, people are more and more interested in wireless networked micro-sensors. EWMN (Embedded wireless Microsensor Network) can be applied to many domains. Based on Bluetooth technology, PMCN data communication technology, embedded Linux, etc., a novel architecture of EWMN is found. Its key technologies are studied in this paper.

Early WSN is mainly applied in mass distribution for sensor nodes in a small area, low wireless transfer speed in sub-network, and short communication distance of

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 312-317, 2005.

single-jump. The new EWMN is mainly used in sparse configuration for sensor nodes, with transfer speed up to 723.2 Kb/s in link layer of the sub-network, and communication distance of single-jump up to 100m.

2 Bluetooth Technologies and the New EWMN Architecture

2.1 Bluetooth Technology and Bluetooth Network

Bluetooth technology is a kind of opening global Radio Frequency (RF) specification between wireless data with pronunciation, and it will enable users to connect to a wide range of computing and telecommunication devices without the need for proprietary cables that often fall short in terms of ease-of-use. The short-range radio frequency connection is the foundation of the application for Bluetooth technology. Bluetooth program is embedded in microchip in order to set up Ad-Hoc wireless connections between the fixed sets and the mobile ones. For example, if Bluetooth technology is introduced in the mobile telephone and laptop computer, the horrible connection cable between the cellular telephone with laptop computer can be thrown away.

Bluetooth piconet and Bluetooth scatternet are both Ad-Hoc radio frequency networks in essence, called by a joint name as Bluetooth Ad-Hoc network, and its equipment is joined as shown in Fig.1. In fig.1, Master and Slave represent the main equipment and subordinative equipment of bluetooth network respectively. The star-like structure network links of point-to-point or one point to multi-points are only established to support single-jump communication; the scatternet is composed of a lot of piconets by interlinking to set up tree-like network structure and other more complicated network link, supporting flexible multi-jump communication. Bluetooth scatternet is a kind of cluster layered Ad-Hoc networks in essence.

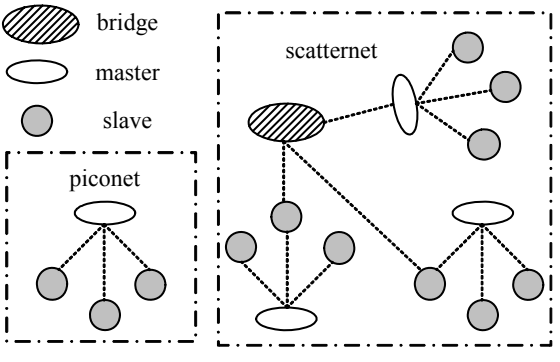


Fig.1. Bluetooth Ad-Hoc network

2.2 The Network Structure of the New EWMN

The network structure of the new EWMN is shown in Fig.1. The structure is divided into three layers: Bluetooth scatternet layer, PMCN layer and Internet layer. Among them, GPRS, GSM and CDMA-1X represent PMCN, and Bluetooth scatternet is a type of layer structure gathered through some interconnected piconets.

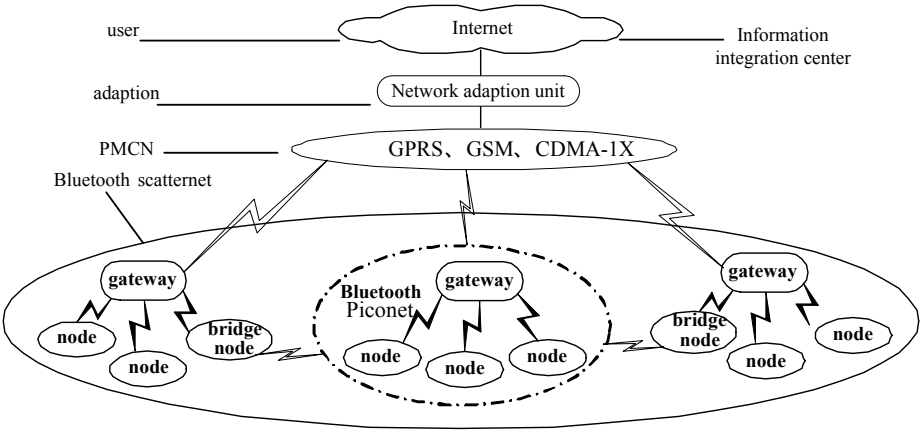


Fig . 2. System structure block of the new EIMN

Any two layers of the new EWMN can interconnect seamlessly. Bluetooth scatternet can interconnect seamlessly with PMCN by Bluetooth Gateway; PMCN can interconnect seamlessly with Internet by Network Adapter Unit [2].

Obviously, the ideal of "unification of three nets with whole wireless" is implemented in the new EWMN: Bluetooth scatternet, PMCN and Internet can interconnect to transfer, exchange, store, deal, and to distribute information obtained by the sensor nodes. Bluetooth short-range communication combines with long-range communication of PMCN to realize wireless whole communication [3, 4].

The main advantages of the layered network structure of the new EWMN may be described as following:

(1) Long-range application and flexible system configuration.

According to the distance of wireless communication and the centralized method to gather the information for sensor node, wireless sensors can be classified into three types: short-range wireless sensor, long-range wireless sensor, whole wireless sensor combined with short-range and long-range together.

(2) Lower equipment cost and communication expenses for the internal node of Bluetooth scatternet to connect Internet.

For a Bluetooth scatternet, only a few sensor nodes are required to have communication capacity with PMCN, which can be called Bluetooth gateway, or be called Bluetooth Sink. If other sensors nodes need to insert PMCN even Internet, Bluetooth gateway can act as agent to realize.

(3) Easy to control and manage in concentrating or dispersing type.

The network is well arranged. Through authorization, authentication and encrypt can be setup.

3 Functions and Application of EWMN

(1) Functions

The basic functions of EWMN are to perceive, gather, deal with, and release the apperceived information. Theoretically, a great deal of accurate and reliable information can be obtained at any time, or in any place and under any environmental condition in EWMN. Multi-jump Ad-Hoc self-organization network can be set up in the sub-network, namely Bluetooth scatternet, in EWMN. Therefore, the operating system such as uClinux can be embedded in Bluetooth sensor node. And it has better retractility in task scheduling and management. Besides the traditional data information, such as temperature, humidity, pressure etc., the information of the picture and radio-orientation information, such as GPS information, is stressed in EWMN. In the aspect of wireless information releasing, the radio is adopted in Internet connection.

(2) Application

While the sub-network is established by short-range radio technologies of Bluetooth and IEEE802.11, according to the geography location of sub-network, the typical application of EWMN can be classified into four types: ① sub-network node location is settled and known; ②sub-network node location is settled but unknown; ③ there is no logical location information and the motion ability is limited inside sub-network; ④ there is no logical location information and the motion ability is better inside sub-network.

The application characteristics of EWMN can be described in three aspects: ① environmental parameter monitoring, safety monitoring, transportation monitoring and orientation tracking in local area; ② the sensor nodes are deployed sparsely; ③ the movement is limited within the sub-network.

4 Main Basic Factors of the New EWMN

(1) Clustering layered structure of Bluetooth scatternet and multi-jump Ad-Hoc self-organized network

Clustering structure layer and multi-jump Ad-Hoc self-organized network are essential features for Bluetooth scatternet. Every piconet is composed of several nearby Bluetooth sensor nodes, and the enabled node of each piconet includes a main equipment node and seven subordinative equipments at the most. The piconet is the first arrangement of the scatternet. The main equipment node of each piconet and the bridge node form the second arrangement of the scatternet.

(2) The senior protocol of Bluetooth scatternet

Though people do their best to study the past protocols of Ad-Hoc wireless network, but the protocols are not suitable for WSN sub-network up to now. The Existing senior protocol profile in Bluetooth technology specification may be reconstructed to apply to Bluetooth scatternet in the new EWMN structure, such as Service Discovery Application Profile, Generic Object Exchange Profile, Object Push Profile, File Transfer Profile, and Synchronization Profile, etc.

(3) Basic equipment

The biggest hardware disposition of the Bluetooth gateway is shown in Fig. 3.

The hardware configuration of Bluetooth sensor node is tighter than Bluetooth gateway. Except the Bluetooth module, sensitive device, embedded microprocessor, memory and power, the other hardware in Fig. 3 can be chosen to use in need.

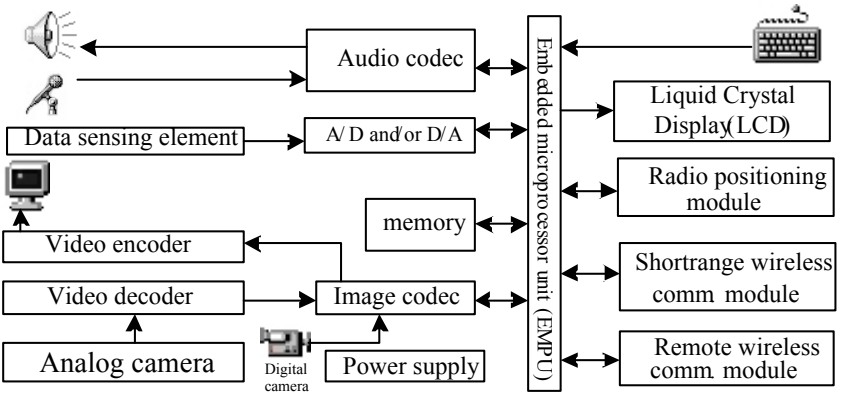


Fig. 3. Maximum hardware configuration of Bluetooth Gateway

(4) Embedded Linux operating system for Bluetooth sensor node

The operating system of Bluetooth sensor node (including Bluetooth gateway) is embedded Linux. Embedded Linux inherits a lot of good characteristics of Linux, for instance, source code opening, small and citable kernel, powerful network function and many kinds of processors supporting, etc. At present, relative perfect embedded Linux includes uClinux, RT-Linux and Embedix, etc. The two embedded architectures are shown as Fig. 4.

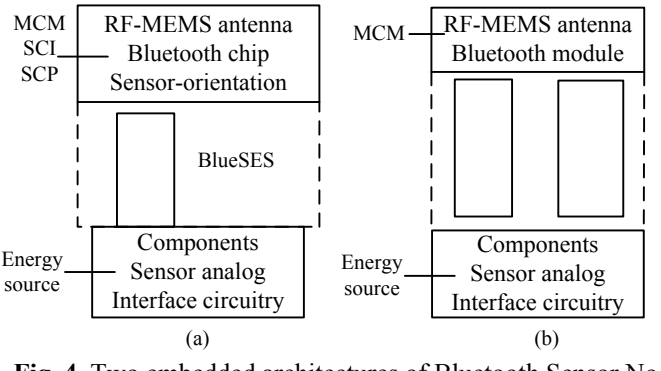


Fig. 4. Two embedded architectures of Bluetooth Sensor Nodes

(5) The information of the multimedia and radio orient obtaining, processing and transmitting

While local safety monitoring, the multimedia information, especially the function of picture information are more outstanding, the corresponding data increase notably. Because the speed of Bluetooth communication and PMCN communication is limited, the multimedia information must be compressed in order to guarantee transfer in time. The key to solve these problems is to compress the picture.

(6) Others

SNMP is adopted in controlling and managing the Bluetooth scatternet, and the information integration center controls and manages the end-users. The existing security mechanism of Bluetooth technology can be improved, and the security mechanism of VPN can be used for reference [5].

5 Conclusions

Embedded Wireless Mirosensor Network has extensive application prospect, and its structure must satisfy specialty of wireless network. The new EWMN proposed in the paper is notably different from the general typical wireless mirosensor network, its characteristics can be summarized as: unification of three nets with whole wireless, Bluetooth scatternet layered, multi-jump Ad-Hoc self-organized, multimedia, orientation, multi-ply Internet inserting, etc. And its main advantages include: long-range application and flexible system configuration; lower equipment cost and communication expenses for the internal node of Bluetooth scatternet to connect Internet; and easy to control and manage in concentrating or dispersing type.

WRIM-GIRS (embedded Wireless network Remote Image Monitoring and GPS Information Receiving System) is an application for the new EWMN. We will study associated hardware fabrication, protocols and algorithm implementations further.

Reference

1. Ravindran V; Varadan V K.: Implementation of local area wireless sensor networks using Bluetooth. SPIE Proceedings, San Diego, CA, USA (2002) 258-265
2. Tilak S.; Abu-Ghazaleh NB; Heinzelman W.: A taxonomy of wireless microsensor network models. Mobile Computing and Communications Review, Vol. 1. (2002) 1-8
3. Bordim, Jacir Luiz; Nakano, Koji; Shen, Hong: Sorting on single-channel wireless sensor netwoeks. International Journal of Foundations of Computer Science, Vol. 14. (2003) 391-403
4. Akyildiz L.F.; Su W.; Sankarasubramaniam Y.; Cayirci E.: Wireless sensor network: A survey. Computer Networks, Vol. 38. (2002) 393-422
5. Perrig, Adrian; Stankovic, John; Wagner, David: Security in wireless sensor networks. Communication of the ACM, Vol. 47. (2004) 41-46

A New Gradient-Based Routing Protocol in Wireless Sensor Networks

Li Xia, Xi Chen, and Xiaohong Guan

Department of Automation, Tsinghua University, Beijing, 100084, China
xiali98@mails.tsinghua.edu.cn
http://www.sensornetwork.net

Abstract. A new gradient-based routing protocol is proposed in this paper. It takes into account the minimum hop count and remaining energy of each node while relaying data from source node to the sink. The optimal routes can be established autonomously with our protocol. A simple acknowledgement scheme, which can be implemented without extra overheads, is proposed. Our protocol also employs data aggregation to save transmission energy. To handle the frequent change of the topology of the network, one scheme for topology update is provided. At last, simulation results illustrate the effect of system parameters on the protocol performance.

1 Introduction

Wireless sensor network is a new technology and it will have significant impacts on human's future life [1∼3]. It consists of a large number of sensor nodes densely deployed in an area of interest. It has a wide range of applications such as military sensing, physical security, environment monitoring, traffic surveillance and etc. A system of networked sensors can detect and track threats and be used for weapon targeting and area denial. As sensor nodes are limited in power, computational capacities, and memory, sensor networks in general pose considerable technical problems in data processing, communication, and sensor management. One fundamental problem is communication protocol. Routing algorithm is one important research topic in wireless sensor network. After sensor nodes gathered the data of circumstance, such data need to be transmitted from the from source nodes to the sink node. Due to limited energy, source node usually cannot send the data to the sink directly. The data need to be relayed by medium sensor nodes. There may be many routes from the source to the sink. Routing is to find the right route. When designing a routing protocol, we need take into account node's energy efficiency (or the lifetime of the sensor network), and possible change of network's topology due to the failure of nodes or various other reasons.

There are several routing protocols proposed in sensor networks. However none of them is always perfect for every situation. For example, LEACH is a good one in many situations [4]. In LEACH, the cluster head node can aggregate

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 318–325, 2005.

the data of its cluster member nodes so that greatly reduce packet transmission in sensor network. However, when the sensor nodes measure different physical phenomenon, this scheme is no longer effective. Moreover, in LEACH algorithm the cluster head nodes directly communicate with the sink node. The head node will consume much more energy when it is far away from the sink node. And the head node may run out of its energy soon.

As we know, the relationship between wireless communication energy consumption E and transmission distance d is: $E \propto d^k$, where k is usually $2 \sim 4$. Hence short distance multiple hop communication is preferable to long distance direct communication in senor network. In this paper, we propose a short distance multiple hop routing protocol. It is derived from the minimum hop count approach. In the traditional minimum hop count algorithm, hop count is the only metric, which measures the quality of route. In our protocol, we not only consider the hop count but also adopt the remaining energy of each node as the metrics of the Quality of Service (QoS) of the link. The goal is to prolong the network's lifetime by optimizing the energy consumption.

In the route setup stage, when one node receives the setup message, it waits for a short time T_{wait} for messages with better metric, which may arrive during this period. When T_{wait} expires, the node rebroadcasts the message with the best metrics in all messages it has received. By this way, the number of setup messages in the whole network can decrease greatly.

According to the omni-direction property of radio signal, when one node relays a packet to its neighboring nodes, it can hear this packet if its neighbor node rebroadcasts this packet. It makes the node be sure that its neighbor node has received the packet. The rebroadcast packet also serves as an acknowledgement from the neighbor node. In our protocol, we adopt such simple acknowledgement scheme.

Relay nodes are augmented with data aggregation function. When relaying packets, they can aggregate similar packets into one packet. Then send out the new aggregated packet. This scheme is helpful to save energy for data transmission.

At last, by simulation, we explore the effect of system parameters on protocol performance and show that this routing protocol performs well comparing with some other traditional routing protocols.

2 Description of the New Routing Protocol

In wireless sensor networks, since the energy of each sensor node is limited, energy conservation is of the most importance to prolong the lifetime of networks.

The routing protocol we propose in this paper can optimize the transmission energy and equalize the energy consumption of all sensor nodes. It can prolong the operating lifetime of the network. Moreover, it has many other features to improve the routing performance. The schemes and algorithm are described one by one in the following subsections.

2.1 Routing Establishment Algorithm

Routing establishment algorithm is the base of our routing protocol. It aims to establish the cost field and find a minimum cost path from the source node to the sink. The cost metrics include current transmission energy consumption, remaining energy and so on. One similar concept can be found in [5].

Initially, each sensor node sets its own cost metric R to ∞ but the sink node's cost is 0. The sink broadcasts a message containing its own cost metric R_{sink}=0. This message is rebroadcast and updated throughout the network. Suppose node M rebroadcasts this message containing its own cost R_M and node N receives this message. Node N compares its own cost metric R_N with $R_M + C_{M,N}$, where $C_{M,N}$ is the transmission metric between node M and N. If $R_M + C_{M,N}$ is smaller, node N sets its cost R_N to $R_M + C_{M,N}$ and records node M as its relay node in its relay list. Then node N rebroadcasts this message containing R_N to its neighbor nodes. If $R_M + C_{M,N}$ is larger than its own cost metric R_N, node N just discards this message without any updates. This process will continue until the message propagates throughout the network. At last, the cost field of the network is established. Each node can find a minimum cost path back to the sink. This path is also called the gradient of the cost field.

In the minimum hop count routing protocol, the cost metric is the hop count between two nodes. The communication metric $C_{M,N}$ is set to hop count between M and N. The source node will find the minimum hop count path to the sink. In our protocol, we use hop count and remaining energy of each node as the metrics. The cost $C_{M,N}$ is set to $\mathit{hop\text{-}count}/E_N$, where *hop-count* is the hop count between node M and N, while E_N is the remaining energy of node N. This metric can balance the energy consumption of the network. It will prolong the operating lifetime of network. Furthermore, we can use the QoS of the link as one of the metrics of the network. We will discuss this issue in another paper.

Figure1 illustrates a simple example of the procedure of generating minimum cost gradient. The source node has three routes to reach the sink, route1: S–>D–>A, rout2: S–>B and route3: S–>E–>C. At the routing setup stage, the source node will receive three different setup messages. The cost metric of route1 is: 1/40+1/40=1/20. It is the smallest cost in these three routes. So the source node will choose route1 as its optimal route. It records node D as its previous relay node. After a period, the nodes in route1 may have low energy level. In this situation, route2 may be chosen as the good route.

In order to improve the routes' fault tolerance ability, we provide several alternative relay nodes for each node. For example, we provide 3 alternative relay nodes for each node. During the routing establishment stage, the second and third best metric nodes are also recorded as members of relay nodes set. In figure1, the relay nodes table of source node is {D, E, B}. The default relay node is D. If the default route is down, we can use the alternative route quickly. This scheme will save a lot of route rebuilding cost. It is also very easy to implement without extra payments.

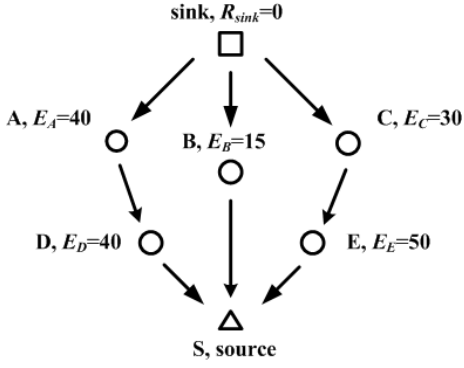


Fig. 1. The procedure of generating minimum cost gradient

2.2 Back-Off Waiting Scheme

When the route setup message propagates through the network, this message may be replicated and rebroadcast many times. Because of the impact of random delay of retransmission, one node is likely to receive and rebroadcast the setup message many times. If so, the node will consume much energy and the explosive setup messages will aggravate the congestion of the network.

We use a back-off waiting scheme to alleviate this effect. When a node receives the setup message whose cost metric is smaller than its own metric, we update it but don't rebroadcast it instantly. It will start a back-off timer and wait for a constant time T_{wait}. During this period, if all the received setup messages' metrics are not better than this one, it rebroadcast the message containing its own metric. If it receives the message whose metric is better than this one, it will update its metric and reset the back-off timer.

This scheme can greatly reduce the number of setup messages in the network. If T_{wait} is large enough, we can see that each node will only rebroadcast the setup message once. It can save a lot of communication energy. The relationship between the number of messages and the parameter T_{wait} is illustrated by simulation in section 3.

2.3 Acknowledgement Scheme

It is well known that wireless communication is frequently influenced by various factors such as multi-path fading, interference and etc. The communication may fail sometimes. On the other hand, sensor nodes usually turn off to save energy. It is possible that the relay nodes are power off when the data packets pass them. So when the packets are relayed from source nodes to the sink, we need to consider the situation when the relay nodes fail to receive the packets.

We propose a simple acknowledgement scheme to handle this problem. This scheme is easy to implement in the sensor networks. It uses the omni-directional radio signal property to acknowledge a received packet. When the relay node

receives the packet and relays it, the sending node will also receive this relayed packet. If the sending node didn't receive it, it means the packet is possibly lost because the relay node is power off or the radio channel is degraded. The sending node will then degrade this relay node's metric and choose another alternative relay node to transfer the packet.

This acknowledgement scheme can improve the success rate of packet transmission. It only brings a little extra payment caused by the sending node's listening to the relayed packets. We can use this scheme to improve the routes' QoS a lot with a small extra energy consumption.

2.4 Routing Update Scheme

In the sensor network, some nodes may be mobile and can move around in the area of interest. Moreover, in order to save energy, sensor nodes may turn off from time to time. Hence, the topology of the network may change frequently. On the other hand, the medium relay nodes may consume energy more quickly than other nodes. Their energy level drops much faster and, after a certain period, their energy level may become too low to take charge of relaying packets. We need a timely scheme to update the routes.

If one sensor node gets ready for turning off, it needs to inform its neighbor nodes. If the neighbor node's relay list includes this node, this node will be indicated as an inactive node in the list. When this node turns on, it also needs to announce this event so that its neighbor nodes can mark it as an active node in the relay node list.

If a node does not know its relay node has turned off, when it sends packet to the relay node, it will not receive any acknowledgement. In this situation, it updates its relay node list and marks this node inactive until receiving the active announcement from the relay node.

When a node moves to a new area, the node needs to update its relay node list. It sends out a request message to ask for its neighbor nodes' cost metric and chooses the node with the smallest metric as its relay node and then updates its relay node list.

Because relay nodes consume their energy more rapidly, the entire routes need to be updated periodically. Such update can be done by following the same procedure as the routing establishment scheme in section 2.1, i.e. the sink broadcasts the setup message and rebuilds the whole routes.

2.5 Data Aggregation

In the sensor network, when an event of interest occurs, the nearby sensor nodes can detect it and send the related information back to the sink. These data may be similar to each other. If we aggregate these packets, relay nodes can save a lot of transmission energy. Hence data aggregation is useful for the sensor network.

In our routing protocol, we implement data aggregation function in the medium relay nodes. When the relay nodes receive data packets, it stores them

and waits for other new packets' coming. After a certain waiting time, the relay node aggregates these packets, which are newly received. Moreover, the medium nodes can utilize their computation resources to analyze the received packets, and only transfer useful information to the sink. This scheme can reduce the amount of data to be relayed in order to improve the energy efficiency and prolong the network's lifetime.

3 Simulation Experiments

In simulation experiments, we consider the area of interest as a square of 100 $\times$ 100 m^2, where 900 sensor nodes are randomly scattered in the area. The transmission range of each node is 10 meters. The sink node is at down-left corner (0, 0). When a node sends a message to its neighbor, there will be a transmission delay T_{delay}, which is caused by radio channel interference or slow transmission data rate. We suppose the delay T_{delay} is uniformly distributed in [0, 50ms]. The full battery energy of each node is assumed to be 10,000 units and sending one packet consumes 2 units of energy.

In routing establishment stage, the sink sends out setup message and sensor nodes involve in relaying it to the network. By simulation, we study the relationship between the number of relayed setup messages and the back-off waiting time T_{wait}.

From Figure 2 we can see, when T_{wait} is larger, the total message for setting up the network's routes will be smaller. When T_{wait} is larger than 40ms, the total number of setup message is about 900, i.e., when T_{wait} is large enough, each node only rebroadcasts the setup message once. So the back-off waiting scheme is quite effective for saving the energy consumption when establishing the network's routes. However, it delays the establishment of routes for a while. The size of such delay is basically proportional to T_{wait}, as illustrated in Figure3.

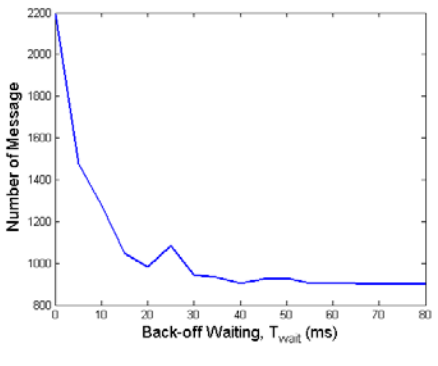


Fig. 2. The relationship between the number of setup messages and T_{wait}

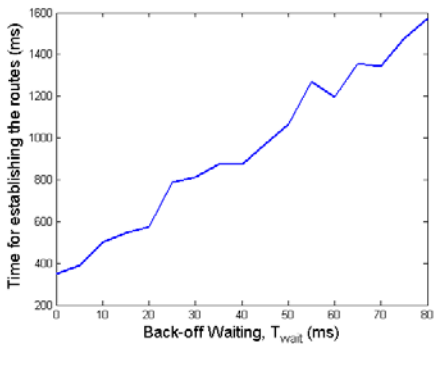


Fig. 3. The relationship between routes establishment time and T_{wait}

In Figure 4, we compare the performance of our new algorithm with the original gradient-based routing algorithm on node exhausting rate. The initial battery energy of each node is a uniform random number between 0 and 10,000. The source packets are produced randomly in the whole network. If the nodes with little energy are still used as relay nodes, they will be exhausted soon. Our routing algorithm can prevent this situation from happening. Simulation results demonstrate that our new algorithm has much better performance than the original one.

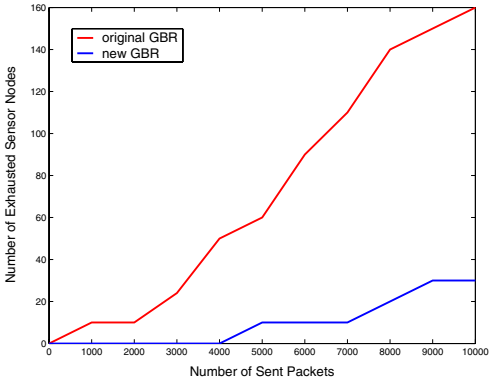


Fig. 4. Exhausted sensor nodes number of two GBR routing algorithms

4 Conclusions

A new gradient-based routing algorithm is proposed in this paper. It optimizes the routes taking into account the remaining energy metric and hop count metric. Back-off waiting scheme is implemented to deal with the explosive message flooding problem in routing establishment stage. A simple and effective acknowledgement scheme is employed, too. Data aggregation in this routing algorithm helps to save energy and prolong the operating lifetime of the whole sensor network.

With these features, our protocol performs better than the traditional one. It prevents long distance direct communication, which is quite energy consuming. When the categories of measured data are different, our protocol is more efficient than other data aggregation protocols, e.g. LEACH algorithm. Our protocol is robust and applicable to the network, whose topology may change periodically or randomly. Also, our protocol is easy for implementation. Our routing protocol can well cooperate with other applications in sensor network. For example, our protocol is good for the application of target tracking because of its adaptability to the changing environments.

At last, simulation results illustrate the efficiency of our routing protocol.

Acknowledgement

This work is partially supported by National Outstanding Young Investigator Grant (6970025), regular NSFC grant (60074012, 60243001, 60274011), 863 High Tech Development Plan (2001AA413910, 2003AA142060) of China, National Key Project of China, Fundamental Research Funds from Tsinghua University, Chinese Scholarship Council and Ministry of Education of China.

References

1. I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci: Wireless sensor network: A survey. Computer Networks **38** March(2002) 393-422
2. H. Gharavi, and S.P. Kumar: Special issue on sensor networks and applications. Proceeding of the IEEE **91** August(2003) 1151-1153
3. C. Chong, and S.P. Kumar: Sensor Networks: Evolution, Opportunities and Challenges. Proceeding of the IEEE **91** August(2003)
4. W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan: Energy-efficient communication protocol for wireless sensor networks. IEEE Proc. Hawaii Int'1. Conf. Sys. Sci. January(2000) 1-10
5. F. Ye, A. Chen, S.W. Lu, and L. Zhang: A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks. Proceedings of Tenth International Conference on Computer Communications and Networks (2001) 304-309

A Sensor Media Access Control Protocol Based on TDMA

Xiaohua Luo, Kougen Zheng, Yunhe Pan, and Zhaohui Wu

College of Computer Science, Zhejiang University
lxhzju@cs.zju.edu.cn

Abstract. Although numerous wireless network protocols are well-studied today, few of them are feasible for super-micro wireless networked sensors with the seriously limited system. In this paper, we present a MAC protocol named "ST-MAC (Sensor Media Access Control protocol based on TDMA)" for wireless networked sensors with the seriously limited resource. Performance analysis in theory and experimental measurements indicate that ST-MAC achieves efficient transmission control with high packet delivery success rate, fair bandwidth allocation and acceptable adaptive ability.

1 Background

Since 1991, although ubiquitous computing still falls short of Mark Weiser's [1] "walk in the woods," advances in hardware, software, and networking over the past dozen years have brought his vision close to technical and economic viability. [2] Embedded platforms are more and more equipped with computational power that allows them to be smart devices with the ability to communicate with their environment. One of the most challenging research fields is wireless sensor networks.

Traditional wireless network protocols are well-studying today. However, in small, low-power and embedded wireless sensor networks, it is quite a different novel region to study. To meet this requirement, we present a MAC protocol named "ST-MAC (Sensor Media Access Control protocol based on TDMA)" for wireless sensors.

2 Related Work

Let's first look through existing mechanisms and the work that has been done for wireless sensor networks before presenting our work.

The IEEE 802.11 [3] standard, which has a handshake series of RTS-CTS-DATA-ACK, does a good job of collision avoidance. But this can be extremely costly. It is unacceptable for sensor networks to apply handshake protocol.

Alec Woo and David Culler in UC Berkeley proposed an adaptive transmission control scheme based on CSMA techniques for sensor networks [4]. But for the lack of detecting on hidden node and time synchronization, the packet delivery success

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 326-332, 2005.

rate is seriously low in special network topology (we will detail it later). In networked sensors, the energy is so precious that we cannot afford lots of transmission failure.

Alec Woo and David Culler did not select TDMA based protocol for the tight requirement of time synchronization and the static network topology. But TDMA is just suitable for smart-home applications. The number of motes in networks is less than 100 generally, and network topology is more static than traditional sensor networks. Above all, TDMA scheme can avoid the hidden node problem furthest. So we can achieve a fairly high packet delivery success rate.

3 ST-MAC Design

3.1 Research Platform

We evaluate the research on Mote, developed at UCB [5]. The main microcontroller is an 8-bit ATMEGA128 running at 16 MHz with 128 Kbytes of flash program memory and 4 Kbytes of system RAM. The RF module consists of a low power Chipcon CC1000 transceiver. It can operate at communication rates up to 76.8 kBaud.

The mote runs on a very small operating system called TinyOS [6], which has been used by over 500 research groups and companies. And based on the platform, we carry on the research about smart-home which meets the need of context awareness and context management.

3.2 Application Assumptions

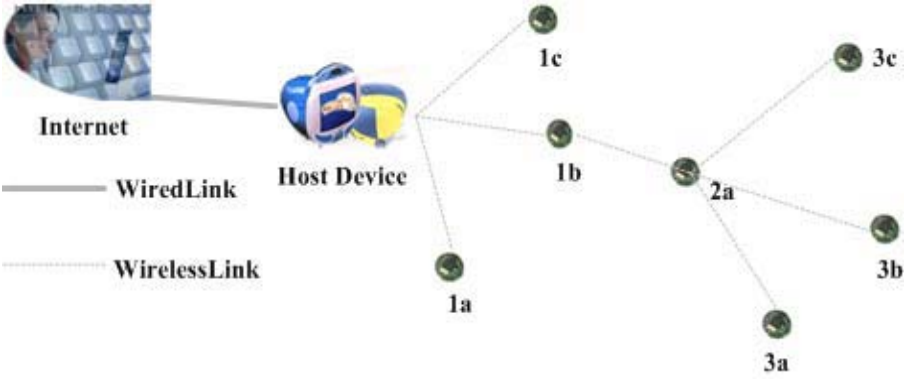


Fig. 1. A sensor network consists of 7 nodes

We assume the application is a data collecting system. As shown in figure 1, several motes form an ad-hoc network to collect environment data periodically and transmit data back to base station. The communication component's default chip rate setting is 38.4 kbps. The data that motes sample periodically, such as temperature or pressure, is relatively small and numerous with a typical size around tens of bytes. With 30 byte packets in Manchester Encoding, the channel capacity can deliver at most 80 packet/s..

Contrasted with motes, the base station can be considered as holding infinite computation power. In general, it can be a smart device equipped with special hardware and software. For the serious absence of computing ability and power supply in motes, the base station should perform tasks as much as possible.

3.3 Design Requirement

Combined with smart-home applications, ST-MAC should be designed to meet some requirements, including efficient transmission control for high packet delivery success rate, fair bandwidth allocation for all nodes in the network and acceptable adaptive ability when the sensor network changes.

Since we choose TDMA scheme, the fair bandwidth allocation is easily achieved. The other points can also be implemented well with special mechanism. ST-MAC should be a collision-free MAC protocol, which uses TDMA scheme to coordinate network transmission. It should automatically detect interfering nodes in a network and, so as to avoid collision.

3.4 Transmission Control

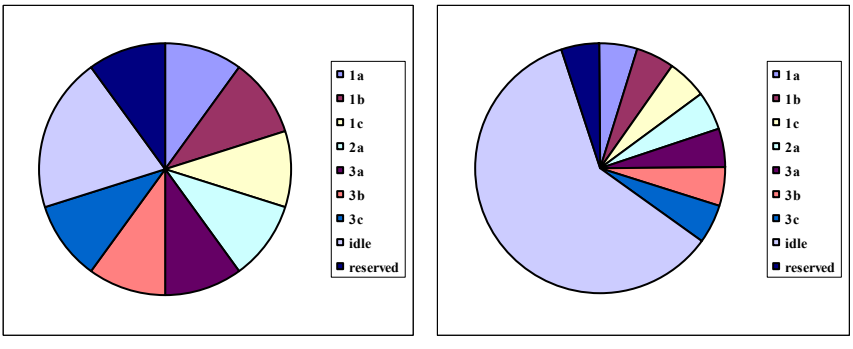


Fig. 2. (A) Configuration with long time slots; (B) Configuration with numerous time slots

As the network topology shown in figure 2, the base station first evaluates the entire network based on the information provided by the motes spread around the environment and calculates a suitable configuration. Then, it broadcasts the command messages to the whole network.

The base station divides one second into 10 time-slices (figure 2(A)), or 20 (figure 2(B)). The last segment of time divisions is conserved for base station to release command messages. Each mote only can transmit packets within its own time slot. If sensor network is relatively static, we can extend the packet size easily with long time slots configuration. If sensor network is changeful, it is advantageous to allocate more idle time slots for new nodes with the second configuration.

For the short range of communication component in mote, the data sampled by node 3a, 3b and 3c must be forwarded by node 2a. In ad-hoc network, it is usual to forward packets when the node lies in the routing path. To save energy, most motes

should power off when time slots are not assigned to them except forwarding nodes. They must listen to the time slots assigned for their children and forward these messages. Commonly, motes should also listen to the last time segment periodically to receive command messages from base station.

3.5 Network Self-Adapting

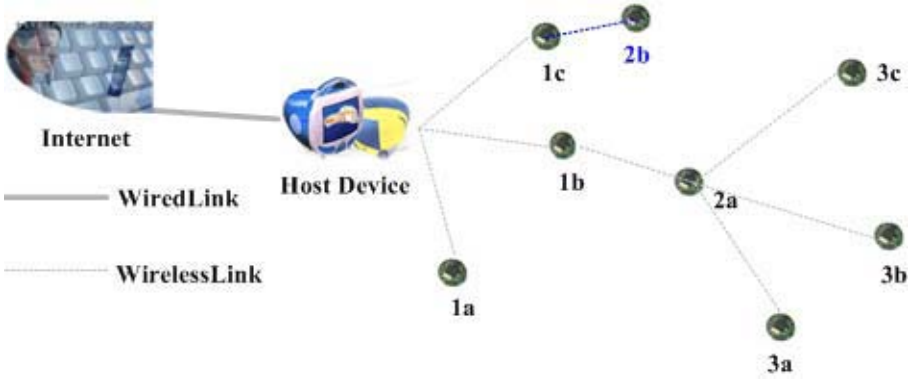


Fig. 3. Network topology reconstructs after a new mote (2b) was added

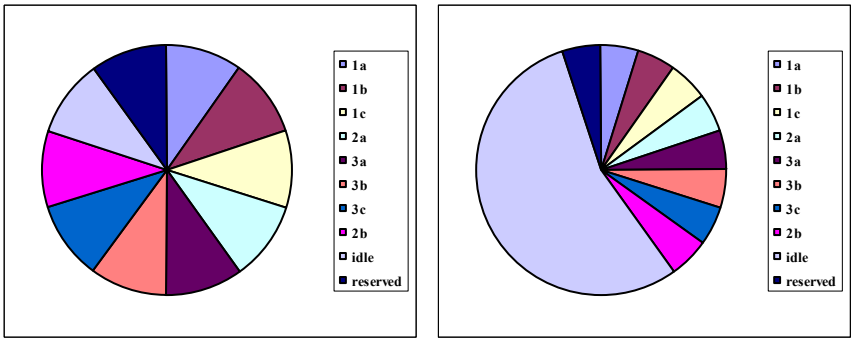


Fig. 4. (A) Configuration with long time slots; (B) Configuration with numerous time slots

The established network configuration is shown in figure 2. If a new mote (2b) is put into the environment, the balance will be disrupted. And the wireless sensor network should be required to be self-adaptive.

At first, the new mote listens to the channel in network and signals messages to claim its presence. Then it keeps silence to listen to command messages from base station. For the disturbing of the new mote, the neighbors beside the new comer will be aware of it, forwarding the discovery to base station. Then the base station will adjust the network configuration (figure 4), and finally set up a new topology shown in figure 3. It is simple to extend the packet length in figure 4(A). Similarly, sensor network can contain more motes in figure 4(B).

If base station has not received messages from one mote for a given time, it will simply delete the mote from the current network topology no matter whether the mote may use up its power or be shielded from the network. There is no competition for channel here. If it is not necessary, base station will not command adjusting channel for it is likely to consume the precious power in motes.

3.6 Time Synchronization

One of the most difficult problems in TDMA implementation is the acquirement of precise time synchronization. In the prior description, we have mentioned that in ST-MAC the base station holds a time slot to release command messages. We can also use it to broadcast synchronous signal when idle. If a mote loses time synchronization for electronic failure, disturbing from outside and so on, it can get synchronization signal as soon as possible. Furthermore, for the constrained number of motes in smart-home applications, the time-slices can be much longer than requirement in theory. So ST-MAC could achieve loose time synchronization in applications.

4 Experiments

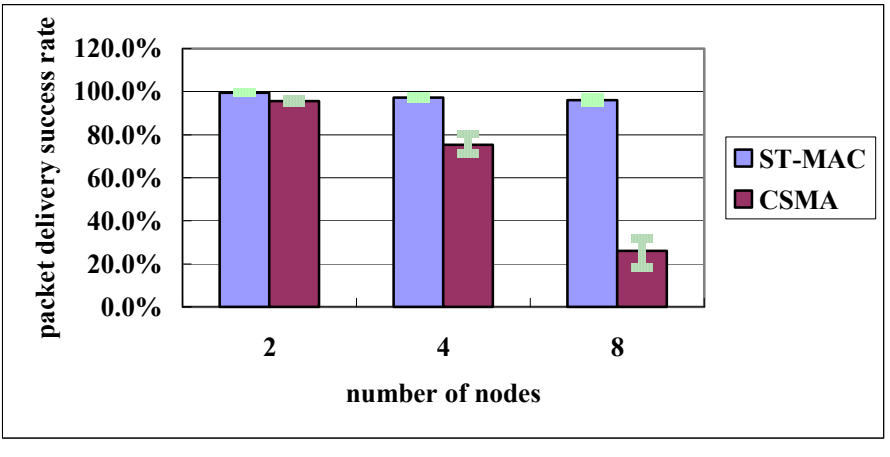


Fig. 5. The comparison of packet delivery success rate with deviation between ST-MAC and the improved CSMA used in UCB

We have practiced ST-MAC on the topology shown in figure 3. Node 1a, 1c and 2a are close to 1b, but can not hear from each other. There is a serious hidden node problem. The same problem also occurs in some other nodes. To measure ST-MAC implementation, we organize the sensor network in 2 nodes (1a, 1c), 4 nodes (1a, 1b, 1c, 2a) and all 8 nodes. In long time slots configuration, we adopt 60 byte packet with 30 byte payload 2 times, set the sample rate to 1 packet/s and analyze all packets centralized in base station during ten minutes. We repeat the experiment ten times.

Figure 5 shows the comparison of packet delivery success rate with deviation between ST-MAC and the improved CSMA used in UC Berkeley. The result indicates that network transmission with CSMA mechanism does not work very well when sensor network expands to 8 motes with special topology and medium payload. In fact, we find that most of the packets delivered by 1b, 2a, 3a, 3b and 3c are seriously interfered or even lost. On the contrary, the performance of ST-MAC is satisfying.

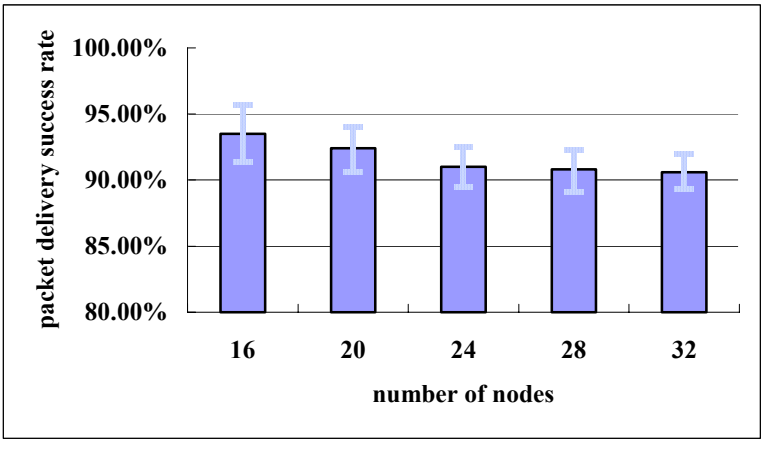


Fig. 6. Simulating result of packet delivery success rate and deviation of ST-MAC

Compared with the improved CSMA scheme applied in UC Berkeley, ST-MAC achieves a fairly high packet delivery success rate besides fair bandwidth allocation. To simulate more large-scale sensor networks, we have improved on the simulator used in UC Berkeley for TDMA scheme and acquired a high success rate stably in any case of the network topology. Figure 6 shows the result.

Though ST-MAC is simple, it works well in sensor network with severe hardware restrictions. The throughput of ST-MAC is also extensible when network enlarged. It is reasonable to consider it a feasible media access control protocol.

5 Conclusion and Future Work

The main purpose of ST-MAC design is to provide reliable packet transmission, and we have achieved it. Though ST-MAC is not as flexible as CSMA implementation, it is still likely to satisfy most applications in tiny smart-home. Despite the shortcomings of the target platform, we are able to demonstrate an available MAC protocol for the prototype sensor network.

ST-MAC is well suited for smart-home applications. But for the strict division of master and slaver, the operation and management of sensor network is highly centralized in base station. It is not very robust. The distributed system is the direction in future to gain high robustness. And we expect to realize the distributed protocol under more powerful platform in near future.

The goals of designing a TDMA based MAC include achieving distributed management and robust operation, providing facilitating flexible energy-saving protocols, supporting nicer network security protocol and secure routing algorithm, enabling resource reservation and QoS aware scheduling, and so on. Of course, there is still a long way to go before these happen.

Acknowledgments

This work is supported by 863 project of China (grant 2003AA1Z2080). We would like to thank our research team, Yingwu Wang, Jinxing Xu, Longlian Li and for all the discussions related to this work.

Reference

1. M. Weiser: The Computer for the 21st Century, Sci.Amer., Sept., 1991.
2. D. Saha, A. Mukherjee, Pervasive Computing: A Paradigm for the 21st Century, *IEEE Computer*, IEEE Computer Society Press, pp. 25-31, March 2003
3. ANSI/IEEE Std 802.11 1999 Edition.
4. Alec Woo, David Culler: A Transmission Control Scheme for Media Access in Sensor Networks, Mobicom 2001, July 2001, Rome.
5. smartdust. http://www.cs.berkeley.edu/~awoo/smartdust/.
6. TinyOS. http://webs.cs.berkeley.edu/tos/.

Clusters Partition and Sensors Configuration for Target Tracking in Wireless Sensor Networks

Yongcai Wang, Dianfei Han, Qianchuan Zhao, Xiaohong Guan, and Dazhong Zheng

Tsinghua University, Beijing, 100084, P.R.China
wangyongcai@mails.tsinghua.edu.cn
http://cfins.au.tsinghua.edu.cn/personalhg/wangyongcai/

Abstract. Decisions on the number of clusters and the sensing radius will effectively affect the quality and energy metrics of WSN (wireless sensor networks) tracking systems. By presenting the mean number of the detectable sensors as a trajectory independent quality metric, an energy-quality optimization model is derived and Pareto based optimization strategy is proposed. The obtained non-bad solutions (Pareto Fronts) can be used to direct the clusters partition and sensors configuration. Comparing with simulation, more than 80% of these Pareto Fronts are coincident with those in experiment results.

1 Introduction

Mobile target tracking, as one of the most important applications of wireless sensor networks (WSN) [1], is widely deployed in military and social applications [3][4][5]. To develop WSN tracking system, *high tracking quality* is the basic requirement from the application level. As in the intrusion detection, demanding for the tracking accuracy is critical. On the other hand, *energy efficiency* is the inherent requirement of the sensors, since these tiny sensors are commonly battery powered, which is difficult to recharge. To satisfy these requirements, cluster-based tracking protocols [4][5][7] are proposed, which give attention to both tracking quality and energy efficiency and are recognized as the most effective solutions for WSN tracking systems. In cluster-based tracking, clusters are formed adaptively, and are scheduled to track the target by predicted activation, which effectively reduce energy cost. Inner cluster data fusion can be implemented by the elected cluster head (CH) to provide more accurate estimation of the target's position. Figure.1 demonstrates a cluster-based tracking scenario. Simulation based investigations has been presented by [3][6]. But because the evaluation of tracking quality depends on the target trajectories, theoretical analysis is difficult for the unpredictable target randomness.

In this paper, we define the quality metric as the mean number of the detectable sensors, which is trajectory independent metric, and we have proved its generality for different detection schemes. With the proposed quality metric, we use convolution to calculate the expression of the overlapping area, and calculate

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 333–338, 2005.

the expectation to derive the theoretical model of the energy and quality metrics in cluster based tracking. The tradeoff of energy and quality is formulated as a two-objective optimization problem and Pareto based solution technique is applied to optimize the two objectives by combined consideration of the number of clusters and sensing radius. Pareto fronts can provide off-line guidance for clusters partition and sensors configuration, which are validated by extensive simulation.

2 Network Model Description

We consider a sensor network consisting of N homogeneous nodes randomly deployed in a L (m) $\times L$ (m) square area. So the sensors destiny denoted by ρ can be calculated as $\rho = N/L^2$. Nodes are assumed to know their own locations by self-localization at initializing phase. The sensing radius of each sensor is s whose value can only be selected in the range [s_small, s_large]. This means that the sensing area of a sensor is a circle. We suppose each sensor has two configured transmission radius: c is used to transmit data within the neighborhood and C is used to communicate with the base station when this sensor works as cluster head, where $C > c$. The configuring sensing and transmission radius are closely based on current sensor devices as Mica, Mica2, etc. During target tracking, sensors switch between two configured states: ***active*** for tracking target, with CPU, sensors and radio on, and ***sleep*** for energy conserving with sensors and radio off. For clustering scheme, cluster heads are selected by VGP algorithm [8] and clusters are partitioned using overlapping cluster activation method [2]. So every cluster is a circle, as demonstrated in Figure 1. If the radius of cluster is R, then an active cluster on average contains $n_c = \pi R^2 \rho$ sensors. Among them, only those within the detectable circle can detect the target. This circle is centered at the target and its radius is s. We denote the number of the active and detectable sensors n_s. Since the overlapping area of active cluster and detectable circle is not a circle, suppose its area is S_d, then $n_s = \rho \cdot S_d$. We assume the cluster members only need one hop to transmit data to the cluster head. According to the energy consumption model [3][7], for one target report, the energy consumed can be formulated as:

$$E = n_c E_{sensor} s^2 + n_s(2 \cdot E_{elec} k + E_{amp} k c^2 + F_{cpu} k) + (E_{elec} k + E_{amp} k C^2) \quad (1)$$

The energy consumption contains n_c times of sensing: $n_c E_{sensor} s^2$; n_s times of transmission, receiving and data processing: $n_s(2E_{elec}k + E_{amp}kc^2 + F_{cpu}k)$; and one time long range communication: $(E_{elec}k + E_{amp}kC^2)$.

3 Quality Metric

In evaluation of cluster based tracking, average tracking deviation is commonly used as quality metric [3][4], which measures how estimated trajectory differs

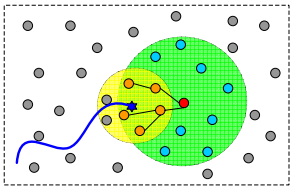


Fig. 1. Cluster based tracking scenario

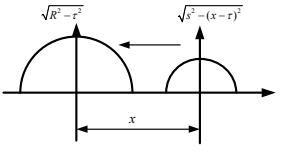


Fig. 2. Convolution of active cluster and detectable region

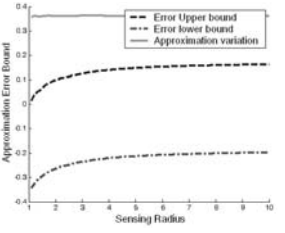


Fig. 3. Approximation error

from the real target one. Average derivation directly describes the tracking performance, but the real target (time, position) pair should be involved in the expression, which makes it depends on the target's trajectory. It is more suitable for the simulation based performance evaluation. For theoretical analysis, the target pattern should be general, and the quality metric should be common for various target patterns. In order to give a proper metric, we perform a close examination of the tracking protocols. According to the functions of the sensors, we discover that although detection methods differ greatly in implementation, they are common in the principle: **the more the better**. The more sensors are involved in tracking target, the more accurate the tracking is. Inspired by this observation, we propose our quality metric as n_s : **the mean number of active and detectable sensors**, which is a quality metric independent to the target's trajectories. The universality of the quality metric can be formulated as theorem 1.

Theorem 1. *If the observation of the sensors are independent and the measurement error is irrelevant with distance, then the variance of estimation error converges at* $O(1/\sqrt{n})$*, where* n *is the number of active sensors that can detect the target.*

Proof of theorem 1 can be referred to [2].

4 Optimization Model

4.1 Derivation of Optimization Model

After introducing the network model and the quality metrics, we derive the optimization model in this section. Since n_s is a linear function of S_d, where S_d is the overlapping area of the active cluster and target detectable area. It is the bridge connecting the parameter R, s and the quality metrics. If we denote x the distance between the cluster head and the target real position, the overlapping area S_d is a function of x. For given R and s, we apply convolution method to calculate the expression of $S_d(x)$. The principle is illustrated in figure 2.

Convolution is used to calculate how the overlapping area varies with x, and the result expression of $S_d(x)$ is:

$$\begin{cases} \pi s^2 & \text{if } R-s>x>0, \\ -f_1\left(\frac{R^2-s^2+x^2}{2x}\right)+f_1(s)+f_2(R)-f_2\left(\frac{-R^2+s^2+x^2}{2x}\right) & \text{if } R+s>x>R-s, \\ 0 & \text{if } otherwise. \end{cases} \tag{2}$$

in which, we write $f_1(\bullet)=(\frac{\bullet}{2}\sqrt{s^2-\bullet^2}+\frac{s^2}{2}\arcsin\frac{\bullet}{s})$, $f_2(\bullet)=(\frac{\bullet}{2}\sqrt{R^2-\bullet^2}+\frac{R^2}{2}\arcsin\frac{\bullet}{R})$ to simplify expression. But the result function is too complex for integration. To simplify $S_d(x)$, we use a piece-wise linear function $\tilde{S}_d(x)$ to approximate $S_d(x)$, where:

$$\begin{cases} \tilde{S}_d(x)=\pi s^2 & \text{if } R-s>x>0, \\ \tilde{S}_d(x)=-\frac{1}{2}\pi s(x-R-s) & \text{if } R+s>x>R-s, \\ \tilde{S}_d(x)=0 & \text{if } otherwise. \end{cases} \tag{3}$$

To check the approximating accuracy, we set s to 1 and vary R and x to calculate the bound of the approximation error over x. More specifically, we define

$$delta_x(R)=\max_x(S_x(R)-\tilde{S}_x(R))-\min_x(S_x(R)-\tilde{S}_x(R)) \tag{4}$$

as approximation error interval. Approximation error interval $delta_x(R)$ and error bounds are shown in figure 3. The interval between the approximation upper bound and lower bound is constant. So we are confident to use $\tilde{S}_d(x)$ to substitute $S_d(x)$ in our following analysis. In $\tilde{S}_d(x)$, the *pdf* of x is unknown. We assume that x is uniformly distributed on the close set $[0,\ R+s]$. By integration the product of the *pdf* of x and $\tilde{S}_d(x)$, the quality metric can be calculated as:

$$n_s=\rho\pi Rs^2/(R+s) \tag{5}$$

So the energy-quality optimization model is derived as:

$$\begin{cases} \text{minimize}\{\ -\ \rho\pi Rs^2/(R+s)\} \\ \text{minimize}\{n_cE_{sensor}s^\beta+(2n_s+1)E_{elec}k+n_s(E_{amp}kc^\alpha+F_{cpu}k)+E_{amp}kC^\alpha\} \end{cases} \tag{6}$$

In (6), minimize$\{\ -\ \rho\pi Rs^2/(R+s)\}$ is converted from maximize$\{\rho\pi Rs^2/(R+s)\}$ to maximize quality metric, and the second item is to minimize energy metric.

4.2 Pareto Based Solution

Since the number of clusters N and sensing radius s can only be selected on discrete points. The model (6) is a two-variable two-objective optimization problem with discrete solution space. We can apply Pareto based optimization method to solve this kind of problem. Figure 6 illustrates the solution method. Every point in the figure stands for a (N, s) pair, which are decision variables. Their x and y coordinates are quality metric and energy consumption respectively, corresponding to equation (6). To maximize tracking quality and minimize energy

consumption, the non-bad solutions should locate on the left down border. They are indicated by circles and called Pareto Fronts, which greatly reduce the solution space. Given application preferences about the energy or quality demands, the constrained optimal (N, s) pair can be searched within the Pareto Fronts. Figure 6 gives an example. The result can be used to guide the clusters partition and sensors configuration.

5 Experimental Evaluation

Extensive simulation is used to validate the optimization model. We simulate the **overlapping cluster tracking protocol** using local prediction as discussed in [4]. Figure 4 and Figure 5 compare the performance metrics of the simulation result with the optimization model. The comparison shows that the derived energy and quality metrics effectively describe the behavior of the tracking system. To further verify the analytical results, we check the Pareto Fronts. Figure 7 shows how the Pareto Fronts obtained from optimization model performs in simulation. More than 90% of the analytical Pareto Fronts are still non-bad solutions in simulation. More experimental results can be referred to [2], which show that for different scale network and different parameter settings, more than 80% of the analytical Pareto Fronts are still not bad solutions in simulation.

6 Conclusions

This paper focuses on theoretical investigation for cluster based target tracking in wireless sensor networks. The mean number of active and detectable sensors is proposed as trajectory independent quality metric, and its generality for different target tracking methods has been proved. A two-objective two-variable optimization model is set up and Pareto-based solution mechanism is applied to direct clusters partition and sensors configuration. The Pareto Fronts will effectively improve the tracking system's performance. For further study, we may apply the analytical results to design tracking systems for smart traffic application, and the experiments will be implemented using MICA2 system.

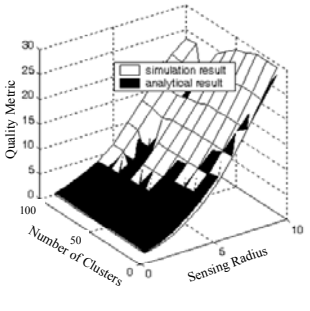


Fig. 4. Comparison of quality metric

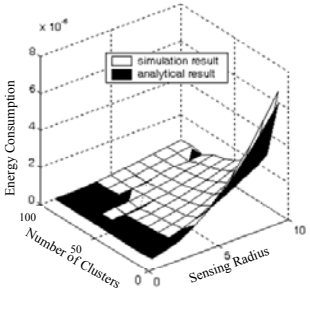


Fig. 5. Comparison of energy metric

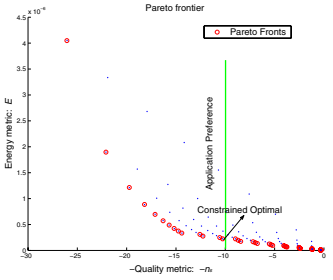


Fig. 6. Pareto optimization with application preference

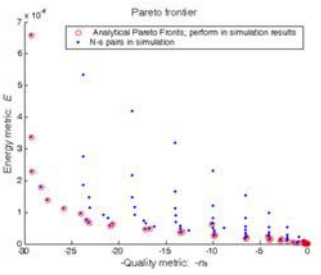


Fig. 7. How analytical Pareto Fronts perform in simulation

Acknowledgement

This work is partially supported by National Outstanding Young Investigator Grant (6970025), regular NSFC grant (60074012, 60243001, 60274011), 863 High Tech Development Plan (2001AA413910, 2003AA142060) of China, National Key Project of China, Fundamental Research Funds from Tsinghua University, Chinese Scholarship Council and Ministry of Education of China.

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer Networks, **38** (2002) 393-422
2. Wang, Y., Han, D., Zhao, Q., Guan, X., Zheng, D.: Energy-quality optimization model for target tracking in wireless sensor networks. Technical report, http://cfins.au.tsinghua.edu.cn/personalhg/wangyongcai/EQmodel.pdf, Automation Dept., Tsinghua Univ, (2004)
3. Pattem, S., Poduri, S., Krishnamachari, B.: Energy-quality tradeoffs for target tracking in wireless sensor networks. Lecture Notes In Computer Science **2634** (2003) 32-46
4. Yang, H., Sikdar, B.: A Protocol for tracking mobile targets using sensor networks. Proceedings of IEEE Workshop on Sensor Network Protocols and Applications, Anchogare, AK, May (2003).
5. Xu, Y., Lee, W.: On localized prediction for power dfficient object tracking in sensor networks. 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03) (2003) 19-22, Providence, Rhode Island, USA
6. Chen, W., Hou, J., Sha, L.: Dynamic clustering for acoustic target tracking in wireless sensor networks. 11th IEEE International Conference on Network Protocols (ICNP'03), Atlanta, Georgia, USA. November 4-7, 2003.
7. Wang, Y., Zhao, Q., Zheng, D.: Energy-driven adaptive clustering data collection protocol in wireless sensor networks. Proceedings of (ICIMA), UESTC, Chengdu, China, Aug. 26-31, (2004)
8. Wang, Y., Zhao,Q., Zheng,D.: "Virtual Grid Point" aided cluster heads election algorithm, submitted to Electronics Letters. (2004)
9. Kaisa, M.:Nonlinear multiobjective optimization, Boston : Kluwer Academic Publishers, (1999)

Enhanced WFQ Algorithm with (*m,k*)-Firm Guarantee

Hongxia Yin, Zhi Wang, and Youxian Sun

National Laboratory of Industrial Control Technology, Zhejiang University, China
hxyin@iipc.zju.edu.cn

Abstract. Weighted Fair Queuing (WFQ) is a good approximation of Generalized Processor Sharing (GPS) service principle, and can provide the delay guarantee when the bandwidth and the burst size are assured. Thus, WFQ can not satisfy the stringent delay requirement of some real-time networked applications with large burst size, such as video transmission. In order to better serve these applications, (*m,k*)-firm guarantee is integrated in QoS architecture. A fluid scheduling algorithm, (*m,k*)-GPS, and its packet approximating scheduling algorithm, (*m,k*)-WF^{2}Q ,are proposed. Dropping some optional packets, they guarantee low delay to flows with large burst size. Moreover, the fairness and computing complexity of (*m,k*)-WF^{2}Q are the same as WF^{2}Q.

1 Introduction

In Internet architectures, scheduling is a main block for providing network-level QoS. To allow a fair share bandwidth among all sessions, Generalized Processor Sharing (GPS) is an ideal scheduling policy in that it provides an exact max-min fair share allocation. GPS is fair in the sense that it allocates the whole outgoing capacity to all backlogged sessions in proportion to their minimum rate requirements. Basically, the algorithm is based on an idealized fluid-flow model. Weighted Fair Queuing (WFQ) is a simple packet-by-packet transmission scheme that is an excellent approximation to GPS even when the packets are of variable length. [1]. However, if the source has a message that is longer than the maximum packet size, it breaks the message up into packets and sends these packets, one at a time, to the network, which leads to burst. For example, when I frame of MPEG video stream is transmitted, large burst comes into being. In the applications mentioned above, the ratio of peak rate to average rate is very large. If guaranteed throughput is close to peak rate, network resource is reduced; if guaranteed throughput is close to average rate, the packet delay becomes very large. Moreover, in overloaded network, WFQ can not guarantee definite delay to flows.

To provide less delay for delay-sensitive applications in ATM, Burst-based Weighted Fair Queuing (BWFQ) is proposed in [2]. However, the delay upper bound under BWFQ is larger than the one under WFQ, meanwhile, the fairness under BWFQ becomes worse. To provide short delay guarantees for

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 339–346, 2005.

real time flows with large burst sizes, (m,k)-WFQ is proposed in [3]. The loss tolerance property of some real time applications such as multimedia streams and networked control systems is utilized in (m,k)-WFQ. However, it is not a guaranteed-rate scheduling algorithm, where the deadline of sessions is taken into account. The assignment of deadline among multi switch nodes is very complex. Furthermore, when some unfriendly applications enter the network, the fairness of (m,k)-WFQ is much worse. In this paper, we also use (m,k)-firm guarantee and propose (m,k)-GPS and (m,k)-WF^{2}Q scheduling algorithm. Dropping some optional packets, they guarantee low delay to flows with large burst size. Moreover, fairness and computing complexity of (m,k)-WF^{2}Q are the same as Worst-case WFQ (WF^{2}Q).

The remainder of this paper is organized as follows. Section 2 provides background of this paper. Section 3 proposes (m,k)-GPS scheduling algorithm, which is used in the fluid systems. Section 4 proposes (m,k)-WF^{2}Q scheduling algorithm, which is used in the packet-by-packet networks. Section 5 shows the simulation results.

2 Background

The classes of GPS, WFQ and WF^{2}Q schedulers are the most general scheduling algorithms and are introduced in detail in [1,4]. Therefore, we omit the introduction about them. In this section, we overview the (m,k)-firm guarantee and define the notations throughout the paper.

2.1 Notations

We will use following notations in explaining all scheduling algorithms mentioned in this paper.

P_i^j: The jth packet to arrival at queue i.
L_i^j: The packet size of P_i^j.
a_i^j: The arrival time of P_i^j.
$d_{i,S}^j$: The finish time of P_i^j under S scheduler.
$D_{i,S}^j$: The delay bound of session i S scheduler.
ρ: The average rate of session i.
σ: The burst size of session i.
C: The service rate of the server.
g_i: The service rate of session i guaranteed by the server.
$R_i(t)$: The arrival curve of session i.
$R_{i,M}(t)$: The arrival curve of mandatory packets in session i.

2.2 (m,k)-Firm Guarantee and (m,k)-Pattern

To better realize optional misses, the (m,k)-firm guarantee model has been proposed [5]. In the (m,k)-firm guarantee model, a session is said to have a (m,k)-firm guarantee requirement if it is adequate to meet the deadlines of m out of

any k consecutive packets of a session where m and k are two positive integers with $m \leq k$. If $m = k$, the system becomes a non-loss service. On the other hand, under the rate-based scheduling algorithms, the deadline of session i is correlative with the end-to-end delay guarantees, i.e., if the end-to-end delay of a packet is not more than the difference between the deadline and the arrival time, this packet is transmitted successfully. Furthermore, [5] gives the definition of (m,k)-pattern as follows:

The (m_i, k_i)-pattern of session i, denoted by Π_i, is a binary string $\Pi_i = \{\pi_{i1}, \pi_{i2}, \ldots, \pi_{in}\}$ which satisfies the following:

(1) π_{ij} is a mandatory packet if $\pi_{ij} = 1$ or optional if $\pi_{ij} = 0$;

(2) $\sum_{i=1}^{k_i} = m_i$.

In order to guarantee the QoS requirement of every session, the scheduler must transmit all mandatory packets before their deadlines. However, the scheduler can drop several optional packets with some policy. Furthermore, the packet loss ratio of QoS parameters is very critical for setting m_i and k_i. In [6], the (m,k)-pattern choose methods are described in detail.

3 Generalized Processor Sharing Based (m,k)-Firm ((m,k)-GPS)

In this section, the generalized processor sharing based on (m,k)-firm guarantee ((m,k)-GPS) scheduling algorithm is proposed, which is based on fluid model. A packet dropper and a scheduler constitute (m,k)-GPS server, where the scheduling algorithm is GPS. The notations and terminologies of (m,k)-GPS are the same as those of GPS. Furthermore, the fairness and complexity of (m,k)-GPS are the same as GPS.

The policy of packet dropper is described as follows: suppose that the last bit of P_j^i is transmitted at time t. If there is a mandatory packet named as P_i^k in queue i, i.e., all optional packets among P_i^k and P_j^i are dropped in case $a_i^k < d_i^j$.

We suppose that the arrival curve of session i accords with leaky bucket, i.e. $R_i(t) \leq \rho_i t + \sigma_i$. Under (m,k)-GPS, the proving process is described in detail in [7]. The maximal backlog of queue i ($B_{i,max}$) and the delay bound of session i ($D_{i,(m,k)-\mathrm{GPS}}$)with leaky bucket are following:

$$B_{i,\max} = \max\left(B_1^{i,M}, \left\lfloor \frac{\sigma_i}{k_i \cdot L} \right\rfloor \cdot m_i L + \sigma_i'\right) \tag{1}$$

where $B_1^{i,M} = max\left(\left\lfloor \frac{\sigma_i}{k_i \cdot L} \right\rfloor \cdot m_i L + \sigma_i' + m_i L - \frac{m_i L}{g_i}\rho_i, 0\right)$.

$$D_{i,(m,k)-\mathrm{GPS}} = \left\lfloor \frac{\sigma_i}{k_i \cdot L} \right\rfloor \cdot \frac{m_i L}{g_i} + \frac{\sigma_i'}{g_i} + \left(m_i L \cdot \left(\frac{1}{g_i} - \frac{1}{\rho_i}\right) \vee 0\right) + \frac{L_{\max}}{g_i} \tag{2}$$

where $\sigma_i' = \begin{cases} \left(\frac{\sigma_i}{k_i \cdot L} - \left\lfloor \frac{\sigma_i}{k_i \cdot L} \right\rfloor\right) \cdot k_i L & \text{if } \left(\frac{\sigma_i}{k_i \cdot L} - \left\lfloor \frac{\sigma_i}{k_i \cdot L} \right\rfloor\right) \cdot k_i \leq m_i \\ m_i L & \text{otherwise} \end{cases}$

The properties and merits of (m,k)-GPS are described as follows:

- (m,k)-GPS is a work-conserving scheduling algorithm. It means that the server must be busy if there are packets waiting in the system.
- (m,k)-GPS is an adaptive scheduling algorithm and does its best to transmit optional packets. According to the network resource, it can choose the amount of dropping packets automatically. If the shared service rate of session i gets larger, fewer optional packets are dropped.

  Proof. Let $[s,\ t]$ be in a busy time period. Because (m,k)-GPS is a work-conserving scheduler, the cumulative output amount of optional packets of session i in the interval $[s,\ t]$ is $g_i(t\text{-}s) - R_{i,M}(s,\ t)$ at least. Obviously, g_i is larger, so more optional packets are transmitted. When $g_i \geq \rho_i$ and $t \to \infty$ (m,k)-GPS is the same as GPS.

- When $\frac{m_i}{k_i}\rho_i \leq \rho_i < g_i$, the delay bound under (m,k)-GPS is definite. However, the delay bound under GPS is infinite.
- To those network applications with large burst size (σ_i), it is obvious that the delay under (m,k)-GPS is smaller than the one under GPS. The reason is that (m,k)-GPS drops some optional packets of burst.

4 Weighted Fair Queuing Based (m,k)-Firm ((m,k)-WF^{2}Q)

In this section, a simple packet-by-packet transmission scheme named (m,k)-WF^{2}Q is proposed with respect to its corresponding (m,k)-GPS scheme. Like (m,k)-GPS, the (m,k)-WF^{2}Q scheduler also has two functions: policing and scheduling. The scheduler drops some optional packets with the dropping policy, and transmits all eligible packets with the scheduling algorithm.

4.1 The Updates of Two Virtual Finish Times

In order to implement packet-by-packet (m,k)-GPS, virtual time $V(t)$ that tracks the progress of (m,k)-GPS is used, and the definition of $V(t)$ is described in [1]. Then denote the virtual times at which this packet completes service as the virtual finish time. Furthermore, denote the virtual finish time of the head packet of the queue i as F_i and the virtual finish time of the last transmitted packet of session i as $\hat{F}_i$. Table 1 shows the updates of F_i and $\hat{F}_i$.

4.2 Policy of Dropping Packets

The dropper under (m,k)-WF^{2}Q is more complicated than under (m,k)-GPS. Because packets can depart much earlier in the (m,k)-WF^{2}Q system than in the (m,k)-GPS system. It is possible that the next packet to be transmitted does not arrival when a packet departs under (m,k)-WF^{2}Q. Therefore, if one of the two following conditions is satisfied, some optional packets are dropped.

Table 1. The updates of F_i and $\hat{F}_i$

The updates of F_i and $\hat{F}_i$	Conditions
$F_i = \max\left(V\left(a_i^a\right), \hat{F}_i\right) + L_i^a/g_i$	P_i^a arrives and queue i is empty
$F_i = \hat{F}_i + L_i^n/g_i$	P_i^l leaves and queue i is not empty
$\hat{F}_i = \hat{F}_i + L_i^l/g_i$	P_i^l leaves

Note: Suppose that $P_i^n$is the head packet of queue i after P_i^l leaves. Define $F_i(0) = \hat{F}_i(0) = 0$ for all i.

- Suppose that P_i^m arrives at the queue of session i. If P_i^m is a mandatory packet and the head packet of queue i is an optional packet and $V\left(a_i^m\right) \leq \hat{F}_i$, the scheduler drops optional packets until the head packet of queue i is a mandatory packet.
- Suppose that a packet departs from the queue i. If there are some mandatory packets in that queue, let the closest packet to the head of the queue be P_i^n. Furthermore, if $V\left(a_i^m\right) \leq \hat{F}_i$ all optional packets before P_i^n are dropped.

4.3 Policy of Transmitting Packets

In order to accurately track the progress of (m,k)-GPS and avoid transmitting some optional packets dropped in the (m,k)-GPS system, (m,k)-WF^{2}Q adopts SEFF (smallest eligible virtual finish time first) policy to service packets [5]. When the server chooses the next packet for transmission at time t, it selects among the set of packets that have started (and possibly finished) receiving service in the corresponding (m,k)-GPS system at time t. And the head packet of session with the smallest virtual finish time is adopted by WF^{2}Q.

4.4 Consistency of (m,k)-GPS and (m,k)-WF^{2}Q

In order to correctly track the progress of (m,k)-GPS, the optional packets transmitted under (m,k)-WF^{2}Q must be the same as under (m,k)-GPS. The following theorem proves the consistency of (m,k)-GPS and (m,k)-WF^{2}Q.

Theorem 1. *For optional packets, (m,k)-WF^{2}Q only transmits those packets that are transmitted under (m,k)-GPS, i.e., optional packets transmitted under (m,k)-WF^{2}Q coincide with ones under (m,k)-GPS.*

Proof. For this theorem, the prove needs two cases.

- (m,k)-WF^{2}Q transmits all optional packets transmitted by (m,k)-GPS. Under (m,k)-GPS, we suppose that P_i^k is an optional packet and is transmitted. Furthermore, we have the following definitions under (m,k)-GPS:
  - Let P_i^h be the last transmitted mandatory packet before P_i^k is transmitted;

- Let P_i^j be the last transmitted packet before P_i^k is transmitted (If P_i^j is a mandatory packet, P_i^j and P_i^h are the same packet).
- Let P_i^l be the first transmitted mandatory packet after P_i^k is transmitted. Considering the above-mentioned assumptions and the definition of (*m,k*)-GPS scheduling algorithm, we have $a_i^l > d_i^j \geq d_i^h$ Because $V(t)$ is an increasing function in a busy period, we also have $V\left(a_i^l\right) > V\left(d_i^j\right) \geq V\left(d_i^h\right)$ .From the policy of dropping packet, we have $V\left(a_i^l\right) > V\left(d_i^h\right)$.Thus, P_i^l can not be dropped by (*m,k*)-WF^{2}Q scheduler. Similarly, $V\left(a_i^l\right) > V\left(d_i^j\right)$ .Thus, P_i^k can not be dropped by (*m,k*)-WF^{2}Q scheduler.

– (*m,k*)-WF^{2}Q does not transmit the optional packets dropped by (*m,k*)-GPS scheduler.

We suppose that P_i^k is an optional packet and is transmitted. Furthermore, we have the same definitions under (*m,k*)-GPS. Under (*m,k*)-WF^{2}Q, one of the following conditions must be satisfied in order to transmit P_i^k

- P_i^l does not arrive until the server has finished transmitting P_i^l.
- P_i^l has arrived when the server finishes transmitting P_i^l. However, $V\left(a_i^l\right) > V\left(d_i^j\right) = \hat{F}_i$.

  If the condition is satisfied, we have $V\left(a_i^l\right) > V\left(d_i^j\right) \geq V_i\left(d_i^h\right)$.Then, $a_i^l > d_i^j \geq d_i^h$. Thus, all optional packets among P_i^h and P_i^l are transmitted by (*m,k*)-GPS.

4.5 Properties of (*m,k*)-WF^{2}Q

From the above analysis, we know that (*m,k*)-GPS ((*m,k*)-WF^{2}Q) is equivalent to the scheduling structure consisting of two components, a packet dropper and a GPS (WF^{2}Q). Furthermore, from Theorem 1, we know that the dropping policy of (*m,k*)-WF^{2}Q coincide with that of (*m,k*)-GPS. Therefore, we can have the same fairness of (*m,k*)-WF^{2}Q as WF^{2}Q, i.e.,

$$C_{i,(m,k)-WF^2Q} = \frac{L_{i,\max}}{g_i} - \frac{L_{i,\max}}{C} + \frac{L_{\max}}{C} \tag{3}$$

where $C_{i,(m,k)-WF^2Q}$ is the Worst-case Fair Index for session i at server (*m,k*)-WF^{2}Q, and $L_{i,\max}$ is the maximal packet size of session i.

In the same way, we know the computing complexity of (*m,k*)-WF^{2}Q is $O\left(\log\left(N\right)\right)$. For the delay bound of (*m,k*)-WF^{2}Q, we have

$$D_{i,(m,k)-WF^2Q} < D_{i,(m,k)-GPS} + \frac{L_{\max}}{C} \tag{4}$$

5 Performance Study

In this simulation, there are three streams sharing a 10Mbps link, and all packets have the same size of 1024 Bytes. The packet inter-arrival time is uniformly distributed in the range $[0.5 * interval, 1.5 * interval]$, where interval is the average inter-arrival time. Table 2 shows the simulation parameters. Fig.1 and 2 show

Table 2. Simulation Parameters in this simulation

Stream	Interval	(m,k)-pattern	Weights
Stream1	0.125	M	0.0064
Stream2	0.004	$MOMOM$	0.0064
Stream3	0.001	M	0.7936

that the delays of streams using (m,k)-WF^{2}Q are smaller than ones using WFQ. The reason is that some optional packets are dropped. Thus, A conclusion can be made that the delays of burst sessions can be reduced using (m,k)-WF^{2}Q.

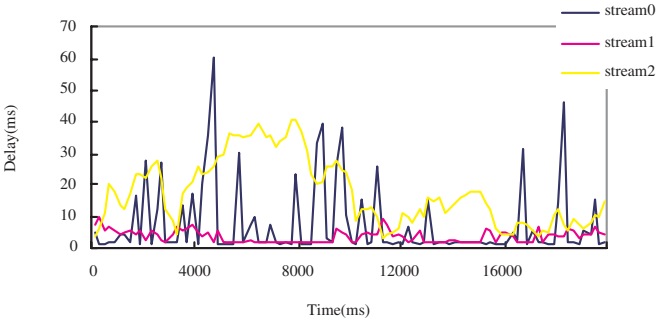


Fig. 1. Delays of streams under (m,k)-WF^{2}Q

6 Conclusions

In this paper, (m,k)-firm guarantee is integrated in QoS architecture, and each packet of sessions is mandatorily or optionally marked at first. Furthermore, (m,k)-WF^{2}Q server is proposed, and it consists of a packet dropper and a scheduler. In the (m,k)-WF^{2}Q system, the delay bound of the session with larger burst size or larger transmission rate is smaller than in the WFQ system. Moreover, because of adopting SEFF policy, (m,k)-WF^{2}Q is more fair than WFQ. Simulation results show that (m,k)-WF^{2}Q is a fair queuing scheduling and (m,k)-WF^{2}Q has

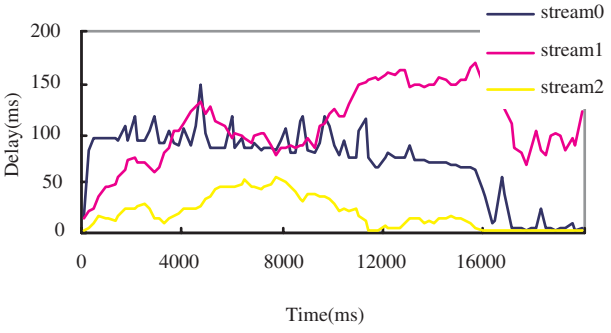


Fig. 2. Delays of streams under WFQ

the same feasibility as WFQ. Therefore, (m,k)-WF^{2}Q is a flexible scheduling for real-time applications.

References

1. Parekh, A. and Gallager, R.: A generalized processor sharing approach to flow control-The single node case. ACM/IEEE Transactions on Networking. 6 (1993) 344-357
2. Chronopoulos, A.T., Yaprak, C.Tang, E.: An Efficient ATM Network Switch Scheduling. IEEE Transactions on Broadcasting, 9 (2003) 110-117
3. Koubaa, A. and Song, Y.Q.: Loss-Tolerant QoS using Firm Constraints in Guaranteed Rate Networks. IEEE RTAS'04, (2004) 526-535
4. Bennett, J.C.R. and Zhang, H.: WF2Q: Worst-case fair weighted fair queuing. IEEE INFOCOM'96. (1996) 120-128
5. Hamdaoui, M. and Ramanathan, P.: A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines. IEEE Transactions on Computers. 12 (1995) 1443-1451
6. Lindsay, W. and Ramanathan, P.: DBP-M: A technique for meeting the end-to-end (m,k)-firm guarantee requirements in point-to-point networks. Proceedings of Conference on Local Computer Networks. (1997) 294-303
7. Yin, H.X., Wang, Z., Sun, Y.X.: A weighted Fair Queuing Scheduling Algorithm with (m,k)-Firm Guarantee. IEEE IECON'04. (2004)

Fuzzy and Real-Time Queue Management in Differentiated Services Networks

Mahdi Jalili-Kharaajoo[1], Mohammad Reza Sadri[2], and Farzad Habibipour Roudsari[2]

[1] Young Researchers Club, Islamic Azad University, Tehran, IRAN
mahdijalili@ece.ut.ac.ir
[2] Iran Telecommunication Research Centre, Tehran, IRAN

Abstract. In this paper, a robust active queue management system to secure high utilization as well as bounded delay and loss is designed. Using this system, the network complies with the demands each traffic class set. We use Fuzzy Proportional-Integral-Derivative (PID) controller in which the parameters of PID controllers are tuned based on fuzzy logic. The proposed controller is simple to implement as well as robust due to the nature of fuzzy controller. Simulation results of the proposed control action demonstrate the effectiveness of the controller in providing robust queue management system.

1 Introduction

The rapid growth of the Internet and increasing demand to use the Internet for time-sensitive voice and video applications necessitate the design and utilization of new Internet architectures to include more effective congestion control algorithms in addition to the TCP based congestion control. As a result, the Differentiated Services (Diff-Serv) architecture was proposed [1] to deliver (aggregated) Quality of Service (QoS) in IP networks. Most proposed schemes are developed using intuition and simple nonlinear control designs. These have been demonstrated to be robust in a variety of simulated scenarios [2,3].

In [4,5] a very useful model is developed to tackle the flow control problem in differentiated services architecture, which divides traffic into three basic types of service (in the same spirit as those adopted for the Internet by the IETF Diff-Serv working group, i.e. Premium, Ordinary, and Best Effort). We will apply fuzzy PID controller [6,7] to such a system.

2 Diff-Serv: New Internet Architecture

In [4], a nonlinear dynamic model for TCP flow control is proposed. Since Int-Serv failed to be adopted for widespread use, the Internet Engineering Task Force (IETF) proposes a more evolutionary approach that does not require significant changes to the Internet infrastructure and provids Differentiation of Services (Diff-Serv) [1]. To

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 347-354, 2005.

accomplish this, Diff-Serv uses the type of service (ToS) field bits in the IP header, which are now renamed as ‘‘DS (Differentiated Services) field’’. The functions associated with these bits have also been redefined. The main issue of the Diff-Serv approach is how to standardize a simple set of mechanisms for handling packets with different priorities denoted by the DS field in the IP header. Note that, in Diff-Serv approach, packet classification is performed only at the edges of the network, which reduces the operational complexity in the network core, and makes it more scalable.

On the other hand, no specific measures are taken to assure that the priorities would actually relate to the desired QoS when a packet leaves the edge router. Therefore, the standard Diff-Serv architecture provides only rudimentary QoS, without any quantified guarantees (unlike the ATM case for example). Because of the availability of limited number of bits in the DS field, the Diff-Serv Working Group has defined a small set of building blocks, called per-hop behaviors (PHBs), which are used by the routers to deliver a variation of services. They are encoded in the DS field and they specify the forward behavior each packet expects to receive by the individual routers. The two PHBs being standardized are the Expedited Forwarding (EF), and the Assured Forwarding (AF). The EF PHB specifies a forwarding behavior with a low loss, low latency, low jitter, and assurs bandwidth end-to-end service, and thus indirectly provides some QoS. In order to ensure that every packet marked with EF receives this service, EF requires from every router to allocate an adequate level of forwarding resources so that the rate of incoming EF packets is always less than or equal to the rate at which the router can forward them. This is done through a Service Level Agreement (SLA) during the connection setup. In order to preserve this property on an end-to-end basis, EF requires traffic shaping and reshaping in the network. Although there is no specific method set for this, it will most probably be a leaky-bucket buffering algorithm. The AF PHB group specifies a forwarding behavior in which packets see a very small amount of loss. The AF PHB group provides delivery of IP packets in four independent forwarding classes. Within each AF class, two or three drop preference levels are used to differentiate flows. The idea behind AF is to preferentially drop best-effort packets and packets non-conforming to contract when there is congestion. By limiting the amount of AF traffic in the network and by managing the best-effort traffic appropriately, routers can then ensure low loss behavior to packets marked with the EF PHB.

3 Dynamic Network Model

3.1 Fluid Flow Model

The full description of Fluid model has been presented in [4,5,.8]. Let $x(t)$ be a state variable denoting the ensemble average number in the system in an arbitrary queuing model at time t. Furthermore, let $f_{in}(t)$ and $f_{out}(t)$ be ensemble averages of the flow entering and exiting the system, respectively. $\dot{x}(t) = dx(t)/dt$ can be written as:

$$\dot{x}(t) = f_{in}(t) - f_{out}(t) \tag{1}$$

The above equation has been used in the literature, and is commonly referred to as fluid flow equation [8]. To use this equation in a queuing system, C and λ have been defined as the queue server capacity and average arrival rate respectively. Assuming that the queue capacity is unlimited, $f_{in}(t)$ is just the arrival rate λ. The flow going out of the system, $f_{out}(t)$, can be related to the ensemble average utilization of the queue, $\rho(t)$, by $f_{out}(t)=\rho(t)C$. It is assumed that the utilization of the link, ρ, can be approximated by the function $G(x(t))$, which represents the ensemble average utilization of the link at time t as a function of the state variable. Hence, queue model can be represented by the following nonlinear differential equation:

$$\dot{x}(t) = -CG(x(t)) + \lambda \tag{2}$$

Utilization function, $G(x(t))$, depends on the queuing in the under study system. If statistical data is available, this function can be empirically formulated. This, however, is not the general case and $G(x(t))$ is normally determined by matching the results of steady state queuing theory with (2). M/M/1 has been adopted in many communication network traffics. In this model, input and service rates both have Poisson distribution function. For M/M/1 the state space equation is:

$$\dot{x}(t) = -C\frac{x(t)}{1+x(t)} + \lambda \tag{3}$$

The validity of this model has been verified by a number of researchers. It is noticeable that (3) fits the real model, however there exists some mismatch. In order to concern the uncertainties, (3) can be modified as:

$$\dot{x}(t) = -\rho C\mu(\frac{x(t)}{1+x(t)} + \Delta)C(t) + \lambda \tag{4}$$

where Δ denotes model uncertainties and

$$\|\Delta\|^2 \le \Delta_{max} \tag{5}$$

3.2 System Structure

Consider a router of K input and L output ports handling three differentiated traffic classes mentioned above. At each output port, a controller is employed to handle different classes of traffic flows entering to that port. An example case of the controller is illustrated in Fig. 2. The incoming traffic to the input node includes different classes of traffic. The input node separates each class according to their class identifier tags and forwards the packets to the proper queue. The output port can transmit packets at a maximum rate of C_{server} to destination where

$$C_{server} = C_p + C_r + C_b \tag{6}$$

3.3 Premium Control Strategy

Premium traffic flow needs strict guarantees of delivery. Delay, jitter and packet drops should be kept as small as possible. The queue dynamic model can be as follows:

$$\dot{x}_p(t) = -C_p(t)\frac{x_p(t)}{1 + x_p(t)} + \lambda_p(t) \tag{7}$$

Here, the control goal is to determine $C_p(t)$ at any time and for any arrival rate, $\lambda_p(t)$, in which the queue length, $x_p(t)$, is kept close to a reference value, $x_p^{ref}(t)$, which is determined by the operator or designer. So in (7), $x_p(t)$ is the state to be tracked, $C_p(t)$ is the control signal determined by the congestion controller and $\lambda_p(t)$ is the disturbance.

The objective is to allocate minimum possible capacity for the premium traffic to save extra capacity for other classes of traffic as well as provide a good QoS for premium flows. Note that we confine control signals as

$$0 < C_p(t) < C_{server} \tag{8}$$

In other words, the assigned premium capacity must always be less than the maximum server capacity C_{server}. This constraint can make the controller design more difficult.

3.4 Ordinary Control Strategy

In the case of ordinary traffic flow, there is no limitation on delay and we assume that the sources sending ordinary packets over the network are capable to adjust their rates to the value specified by the bottleneck controller. The queue dynamic model is as follows:

$$\dot{x}_r(t) = -\rho_r C_r(t)\left(\frac{x_r(t)}{1 + x_r(t)} + \Delta\right) + \lambda_r(t - \tau) + \lambda_b(t) \tag{9}$$

The control goal here is to determine $\lambda_r(t)$ at any time and for any allocated capacity $C_r(t)$ so that $x_r(t)$ can be close to a reference value $x_r^{ref}(t)$ given by the operator or designer. Some points here must be taken into consideration:

a) Total arrival rate is

$$\lambda(t) = \lambda_r(t-\tau) + \lambda_b(t) \tag{10}$$

where $\lambda_r(t-\tau)$ is the rate specified by the controller and sent from sources to the bottleneck router, τ denotes the round-trip delay from bottleneck router to ordinary sources and back to the router and $\lambda_b(t)$ is the arrival rate of the background traffic, which is any extra traffic passing the ordinary queue and should be considered as a disturbance in the controller design. We assume that

$$\lambda_b(t) << \lambda_r(t-\tau) \tag{11}$$

b) Δ as like in the premium case, denotes modeling uncertainty in which $\|\Delta\|^2 \leq \Delta_{\max}$.

c) $C_r(t)$ is the remaining capacity, $C_r(t)=C_{server}-C_p(t)$ and should be considered as disturbance, which could be measured from the premium queue. In our controller scheme we will try to decouple the affect of $C_r(t)$ on the state variable $x_r(t)$.

d) Another constraint that makes controller design more challenging is that λ_r is limited to a maximum value, λ_{max}, and no-negative λ_r is allowed, i.e.,

$$0 <= \lambda_r(t) <= \lambda_{max} <= C_{max} \tag{12}$$

3.5 Best-Effort Traffic

As mentioned in the previous section, best effort traffic has the lowest priority and therefore can only use the left capacity unused by Premium and Ordinary traffic flows. So, this class of service is no-controlled.

4 Fuzzy PID Congestion Controller Design

In this section, the fuzzy PID controller is designed as described above. We have made the following assumptions for controller design throughout this paper:

C_{max}=300000 *Packets Per Second*
λ_{max}=280000 *Packets Per Second*

Table 1. Rule base of the premium fuzzy PID controller

R1	If (Xref is VS) and (λ is VS) then (Kp is B) and (Ki is M) and (Kd is S)
R2	If (Xref is S) and (λ is VS) then (Kp is M) and (Ki is M) and (Kd is S)
R3	If (Xref is M) and (λ is VS) then (Kp is M) and (Ki is M) and (Kd is S)
R4	If (Xref is B) and (λ is VS) then (Kp is M) and (Ki is S) and (Kd is S)
R5	If (Xref is VS) and (λ is S) then (Kp is M) and (Ki is B) and (Kd is S)
R6	If (Xref is S) and (λ is S) then (Kp is M) and (Ki is B) and (Kd is S)
R7	If (Xref is M) and (λ is S) then (Kp is M) and (Ki is B) and (Kd is VS)
R8	If (Xref is B) and (λ is S) then (Kp is M) and (Ki is B) and (Kd is VS)
R9	If (Xref is VS) and (λ is M) then (Kp is M) and (Ki is S) and (Kd is S)
R10	If (Xref is S) and (λ is M) then (Kp is B) and (Ki is M) and (Kd is VS)
R11	If (Xref is M) and (λ is M) then (Kp is M) and (Ki is B) and (Kd is VS)
R12	If (Xref is B) and (λ is M) then (Kp is M) and (Ki is B) and (Kd is VS)
R13	If (Xref is S) and (λ is B) then (Kp is B) and (Ki is M) and (Kd is VS)
R14	If (Xref is B) and (λ is B) then (Kp is M) and (Ki is S) and (Kd is VS)

The proper rules are chosen for both of systems regarding to variation of the $x_{ref}(t)$ and disturbance. To obtain the best fuzzy rule base, using the triangle membership functions, the control rules are established by trial and error approach for the premium and ordinary fuzzy PID controllers as Tables 1 and 2. K_p, K_i and K_d are proportional, integral and derivative gains for PID controller, respectively. The variations of these parameters are according to Tables 1 and 2. In Table 3 the range of the parameters is defined. The simulation results are depicted in Figs. 1, 2, and 3 for Premium traffic,

and in Figs. 4, 5 and 6 for Ordinary traffic. $x(t)$ and $x_{ref}(t)$ for Premium and Ordinary traffics are shown in Fig. 1 and Fig. 4, respectively. As it can be seen, the signal can follow its set-point well and good behavior for rising and settling of $x(t)$ is clear in both of them. The input and output rate of Premium buffer are shown in Figs. 2 and 3, respectively. Figs. 5 and 6 show the output and input rate for the ordinary buffer as well.

Table 2. Rule base of the ordinary fuzzy PID controller

R1	If (Xref is VS) and (C-Cp is VS) then (Kp is M) and (Ki is M)
R2	If (Xref is VS) and (C-Cp is S) then (Kp is M) and (Ki is M)
R3	If (Xref is VS) and (C-Cp is M) then (Kp is B) and (Ki is B)
R4	If (Xref is VS) and (C-Cp is B) then (Kp is B) and (Ki is B)
R5	If (Xref is S) and (C-Cp is VS) then (Kp is S) and (Ki is M)
R6	If (Xref is S) and (C-Cp is S) then (Kp is VS) and (Ki is VS)
R7	If (Xref is S) and (C-Cp is M) then (Kp is S) and (Ki is M)
R8	If (Xref is S) and (C-Cp is B) then (Kp is S) and (Ki is M)
R9	If (Xref is M) and (C-Cp is VS) then (Kp is S) and (Ki is M)
R10	If (Xref is M) and (C-Cp is S) then (Kp is VS) and (Ki is VS)
R11	If (Xref is M) and (C-Cp is M) then (Kp is M) and (Ki is M)
R12	If (Xref is M) and (C-Cp is B) then (Kp is M) and (Ki is M)
R13	If (Xref is B) and (C-Cp is VS) then (Kp is S) and (Ki is M)
R14	If (Xref is B) and (C-Cp is S) then (Kp is S) and (Ki is M)
R15	If (Xref is B) and (C-Cp is M) then (Kp is M) and (Ki is M)
R16	If (Xref is B) and (C-Cp is B) then (Kp is M) and (Ki is M)

Table 3. Parameters range of fuzzy rules

	VS (Very Small)	S (Small)	M (Medium)	B (Big)
Xref	[0 50]	[25 100]	[50 250]	[200 1500]
λ & C-Cp	[0 50000]	[40000 110000]	[100000 220000]	[200000 300000]
K_p	[1500 5000]	[6000 10000]	[12000 20000]	[60000 100000]
K_i	[3000 6000]	[12000 20000]	[25000 40000]	[60000 100000]
K_d	[0 0.1]	[0.07 0.3]	------	------

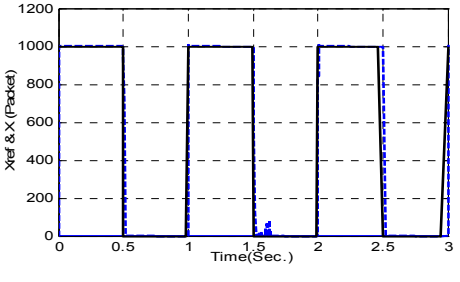


Fig. 1. xpref(t) and xp(t)

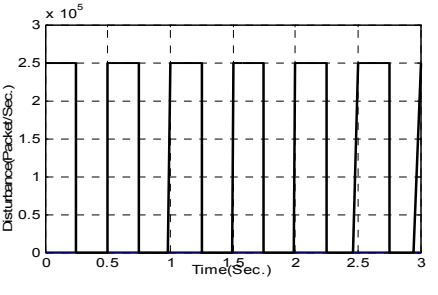


Fig. 2. Input rate of premium buffer (λp(t))

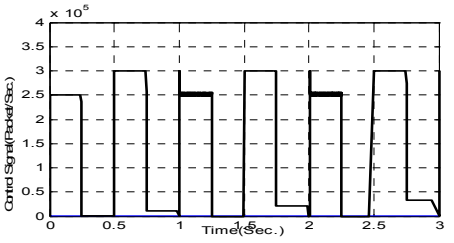


Fig. 3. Output rate of premium buffer($C_p(t)$)

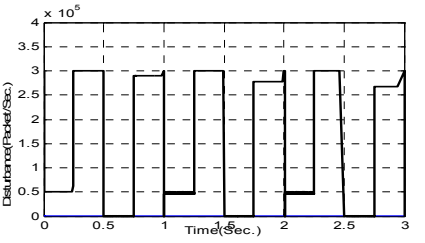


Fig. 5. Output rate of ordinary buffer ($C_o(t)$)

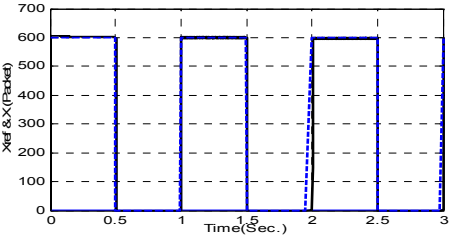


Fig. 4. $x_o^{ref}(t)$ and $x_o(t)$

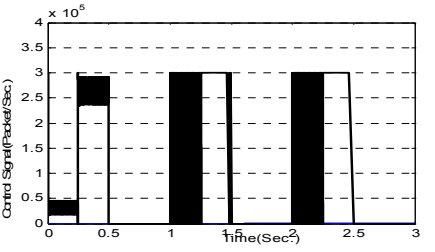


Fig. 6. Input rate of premium buffer ($\lambda_o(t)$)

5 Conclusion

This paper proposes a robust scheme for congestion control based on fuzzy PID control theory, which uses an integrated dynamic congestion control approach in Diff-Ser networks. We divide traffic into three basic types of service (in the same spirit as those adopted for the Internet by the IETF Diff-Serv working group, i.e. Premium, Ordinary, and Best Effort). The controller works in an integrated way with different services and has simple implementation and low computational overhead, as well as featuring a very small set of design constants that can be easily set (tuned) from simple understanding of the system behavior.

References

1. Blake, S., et al., An architecture for Differentiated Services. RFC 2475, 1998.
2. Jain, R., S. Kalyanaraman, R. Goyal, S. Fahmy, R. Viswanathan, ERICA switch algorithm: a complete description, ATM FORUM, AF/96-1172, 1996.
3. Rohrs, C.E., R.A. Berry, S.J. O'Halek, A Control Engineer's Look at ATM Congestion Avoidance, in Proc. IEEE GLOBECOM'95, Singapore, 1995.
4. Pitsillides, P., L. Ioannou, B. Rossides, Congestion Control for Differentiated-Services using Non-Linear Control Theory, in Proc. Sixth IEEE ISCC, Tunisia, pp. 726-733, 2001.
5. Jalili-Kharaajoo, M., B.H. Khazerouni, Robust nonlinear control algorithm applied to congestion control in differentiated services networks, in Proc. IEEE INDIN, Canada, 2003.

6. Chen, Conventional and fuzzy PID controllers: An overview, Int. J. Intelligent Control Systems, 1, pp. 235–246, 1996.
7. Misir, D., H. A. Malki, G. Chen, Design and analysis of a fuzzy proportional-integral-derivative controller, Fuzzy Sets Systems, 79, pp. 297–314, 1996.
8. Jalili-Kharaajoo, M., B.N. Araabi, Application of predictive control algorithm to congestion control in differentiated services networks, LNCS, 3124, 2004.

Issues of Wireless Sensor Network Management[1]

Zhigang Li, Xingshe Zhou, Shining Li, Gang Liu, Kejun Du

School of Computer, Northwestern Polytechnicial University, Xi'an, China, 710072
cnypg@sina.com, zhouxs@nwpu.edu.cn, dtlsn@yahoo.com.cn,
lgslr@163.com, kejundu@hotmail.com

Abstract. In future application environments, wireless sensor networks may comprise tens of thousands of nodes and multiple applications will be executed concurrently. Until now, WSNs and their applications have been developed without considering a management scheme. In this article we discuss the management technologies for WSNs. We identify what is SNM and its necessities. Based on multi-sensor management technologies in data fusion, we propose a feedback management framework for WSNs. Finally, a macro-micro management architecture for hierarchical WSN is also proposed.

1 Introduction

The rapid advances in MEMS and wireless communication technologies have enabled the integration of sensing, actuation, processing and wireless communication capabilities into tiny sensor devices. These sensors can then be deployed in large numbers to self-organize into networks that serve a wide range purposes, including environmental monitoring, infrastructure management, industrial sensing, medical, and military [1].

During the past few years, a lot of research efforts have focused on technologies of networking these tiny sensor devices. Energy efficient MAC, topology control protocols and routing schemes are implemented and evaluated. In addition, methods on how to leverage the distributed computing environment provided by these devices for extracting reliable and timely information from sensing data are also studied [2]. Some prototype applications have been developed and experimented to identify research challenges.

Until now, techniques for WSNs focus on simple data-gathering-style applications, and in most cases they support one application per network, and the network size is small. So WSNs and their applications have been developed without considering a management solution. But in the future, a WSN may comprise tens of thousands of sensor nodes, and multiple applications will be required to be concurrently executed over a single network. For instance, a battlefield surveillance system may have tens of thousands of nodes to simultaneously monitor physical environment, detect biological and chemical attack, track hostile tanks, assess battle damage, and communicate with

[1] This work is partially supported by National Science foundation of China under Grant No.60273086.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 355-361, 2005.

soldiers and other systems. In such dynamic complex systems, WSNs will need to reconfigure and adapt themselves to changes of environment and mission requirements.

In this paper, we focus our attention on the problem of managing WSNs. Recently, several management solutions for WSNs, whose main motivations are to adopt ad hoc network management techniques to WSNs, such as MANNA [3], have been proposed. WSNs are distributed multi-sensor systems connected by wireless ad hoc network. So management of WSNs should combine multi-sensor management techniques for information fusion with network management techniques for wireless ad hoc network. In this article, we implement it in the middleware environment.

The rest of this paper is organized as follows. Section 2 presents our assumptions of WSN. Section 3 discusses the roles and objectives of WSNs management (SNM), and perspective of SNM with the other parts of WSNs systems. Section 4 describes the management architecture of WSNs and we conclude in Section 5.

2 Preliminaries

A. Hierarchical Sensor Network

In recent years, hierarchical structure is widely accepted in designing sensor network because clustering can optimize network performance [4]. In this paper, we consider a heterogeneous multi-hop network which consists of two types of nodes: cluster head and sensor node, as depicted in Fig. 1. Cluster head has two wireless transceivers -- one for inter-cluster communication, and another for intra-cluster communication. It has sufficient energy and can know its location through GPS. While simple sensor node has limited energy and various sensors, it only knows its relative location to cluster head.

B. Deployment and Cluster Forming

The deployment of sensor nodes is random and nodes distribution is the law of a two-dimensional homogeneous Poisson point processes. Heinzelman found the optimal number of clusters scales as $\sqrt{n}$ in designing Leach protocol [4]; Mhatre got similar result in studying minimum cost multi-hop heterogeneous WSN [6]. After random deployment, cluster forming is the construction of Voronoi diagram.

C. Communication in Two Tiers

There is high bandwidth among clusters, so communication among clusters can adopt traditional ad hoc routing protocols. The main objective of collaboration among clusters is providing application QoS. While within a cluster the communication bandwidth is low and routing protocols specially designed for WSN are adopted. The main objective of collaboration in a cluster is energy-efficiency.

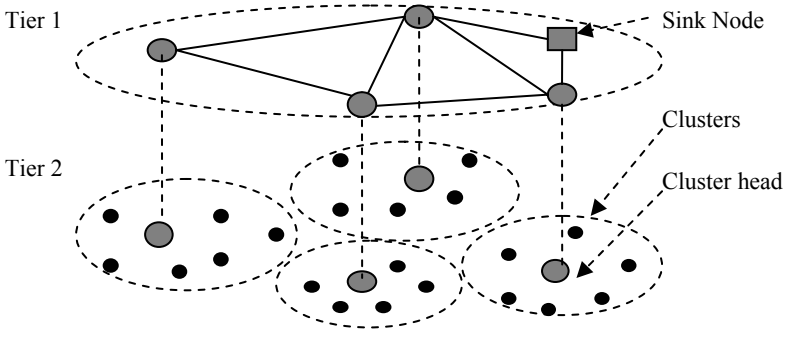


Fig. 1 Hierarchical sensor network

3 Management of Wireless Sensor Network

3.1 What Is SNM (Sensor Network Management)

Referring the definition of SM [4], we describe SNM as a system or process that seeks to manage and coordinate the sensor nodes in a dynamic and uncertain environment, to accomplish specific mission objectives and improve the performance of perception, by using least amount of energy. The word 'manage' gives a sense of control over the sensor nodes; 'coordinate' brings out the efficient use of the resources of sensor network.

Due to application-specific characteristic of WSNs, an important goal of SNM is to optimize the performance of application system. Clearly, there are a wide range of applications for sensor networks with different requirements. Based on the characteristics of these applications, we can classify them as follows:

- Querying parameters of physical environment. For example, a user wants to know the temperature and humidity of room 806, or he wants to query the temperature of which room is above 30℃.
- Detecting some events and assessing influence of events. For example, a user may care about whether area A has poisonous gas and what is the concentration if found.
- Classification or identification. Such as identifying whether the object entering area A is an animal or a tank.
- Tracking moving objects.

These four kinds of applications have different performance metrics, such as detecting probability, identification accuracy, and probability of loss-of-track. Basically, we desire the application performance metric established for optimization to transcend the diversity of sensors and to be analytically/computationally tractable. Further, some system/network performance metrics, such as lifetime and latency, should be taken into account. Energy is a critical resource in WSNs, all operations performed in the network should be energy-efficient. In some applications, the data collected by network may be of no value unless the observer receives it in time. So

the primary objective of the sensor network management is to balance among these performance metrics to get maximized overall performance. In details, SNM seeks to answer the following questions:

- Which tasks are to be performed?
- Which set of sensor nodes is to be allocated to which task?
- How to self-configure the ad hoc network?
- How to coordinate among relevant sensor nodes?

In a nutshell, each sensor in WSNs is fault-prone and detects only limited amount of information, but the sensor network manger can direct sensor nodes in an integrated mode to supply information to the fusion/aggregation process. By doing so, it provides greater content with lower uncertainty than information from a single sensor.

3.2 SNM System in Perspective

The roles and functions of SNM can be understood if we can identify them into different levels based on their functionality. Here, we identify three levels, namely:

- Node layer: this is the lowest level of work in SNM involving individual control of each sensor node such as switching among different modes (sleep/active/idle) of components, and controlling sensors. In wireless sensor networks, due to energy constraint, sensor nodes are always equipped with passive sensors. For passive sensor, the management issues could be unique to each sensor, such as controlling the scan rate and detection threshold.
- Network layer: this is the medium-level SNM. At this level, the SNM focuses more on the management of wireless ad hoc network, including network operation parameters configuration, network connectivity maintenance, topology finding and controlling, moving nodes management, etc.
- Task layer: this can be seen as higher-level SNM. It involves assignment and coordination of sensing tasks, task scheduling, etc.

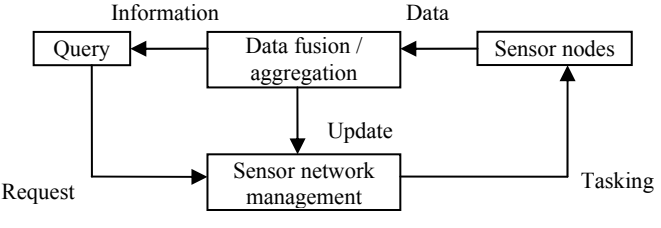


Fig. 2. SNM framework

From multi-sensor management perspective, we can present the relationship of SNM with other parts of sensor network as in Fig. 2. We see that in this framework, when collected sensing data are sent from the sensor nodes to sink node, the data are processed by data fusion system. In this case, low-level fusion is typically performed on data from neighboring sensors before being sent to a next hop node. At the cluster

head (with higher processing capability) or sink node, high level fusion can be performed to extract useful information.

The inputs to the SNM may come from user's requests of specific mission objective through querying. Also, the data fusion system will update the SNM, e.g., update on existing tracks from result information. The states of network and sensor nodes are the third input to SNM. With these inputs, the SNM's role is to optimally manage the sensor nodes. In this framework, sensor network management provides information feedback from data fusion results to sensor node's operations. The feedback is intended to improve the qualities of data collection process.

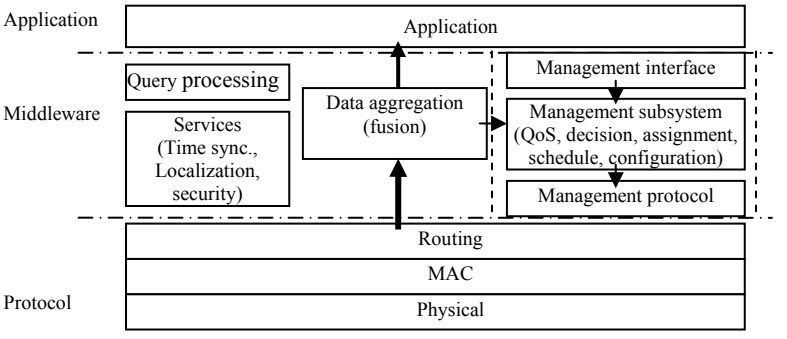


Fig. 3. System architecture including SNM

Recently, much work has targeted the development of middleware specifically designed to meet the challenges of wireless sensor networks [5]. In order to implement SNM, we consider putting it in WSNs middleware environment. Fig. 3 presents the overall system architecture. The right part of middleware is SNM, which comprises management interface, management subsystem and management protocol. In this implementation, security management is moved form SNM and made as a service. The reason is WSNs are employed for diverse set of applications ranging from military battlefield surveillance to home applications and these different applications have different security requirements.

4 WSN Management Architecture

Reference [8] discusses that in dynamic and resource-constrained environments like MANETs, the conventional manager/agent management paradigm will not be efficient. But for a less dynamic and heterogeneous WSN described in section 2, we can still use manager/agent approach.

In WSNs, the architecture of SNM system is related to data collection and communication strategy and the form of data fusion system. Typically there are three types of network management architecture: centralized, distributed, and hierarchical. Hierarchical management architecture can be regarded as a mixture of centralized and distributed architectures. It uses intermediate managers to distribute the manager

tasks. Each intermediate manager has its own subnetwork; it collects and processes information of its subnetwork and passes the information to the upper level manager. According to hierarchical architecture, we proposed a macro-micro architecture for SNM that fits nicely into our WSN model, as depicted in Fig. 4. Macro-manager is in charge of high level strategic decisions about how to best utilize the available sensing resources to achieve the mission objectives and distributes management policies to micro-managers. According to application requirements and information in MIB, macro-manager chooses a set of clusters to participate in data acquisition and allocates tasks to these clusters. It is located at sink node (the network has only one powerful sink node). Micro-manager maintains network topology in a cluster and schedules sensor nodes to best carry out the requests from the macro-manger. Micro-manager is located at cluster head. In conventional hierarchical management architecture, there is no direct communication between intermediate managers. But in our macro-micro architecture, micro-managers collaboratively carry out management operations.

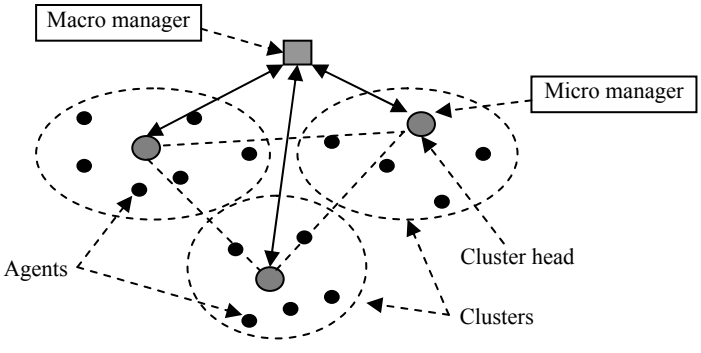


Fig. 4. Macro-micro management architecture

WSN agents are the entities that interface to the actual sensor node being managed. Micro-managers interact with WSN agents to perform management tasks through management protocol. Management information collected from network is represented in a structured manner in MIB. Each element in the network maintains an MIB that has information about its current configuration, operation statistics, and parameters to control operations. Due to the energy constraint, in WSNs it can introduce methods of measurement plus probabilistic forecast to update MIB.

5 Conclusion

Due to the continuing advances in network and application design in WSNs, the development of a sensor network management system is becoming necessary and possible. Because the significant differences between traditional networks and WSNs, a different management solution is required. In this article, based on the techniques of multi-sensor management, we propose a closed-loop feedback management framework and macro-micro management architecture for WSNs. However, to

implement successful network management system for WSNs still needs significant work to resolve many technical issues.

References

1. I. F. Akyildiz, W. Su, Y. Sankarsubramaniam and E. Cayirci, Wireless Sensor Networks: a survey, Computer Networks, Vol. 38, pp. 393-422, Mar. 2002.
2. Sri Kumar, Feng Zhao and David Shepherd, Collaborative Signal and Information Processing in Microsensor Networks, IEEE Signal Processing, Vol. 19, Issue 2, Mar. 2002.
3. Ruiz, L.B., Nogueira J.M., Loureiro A.A.F., MANNA: A Management Architecture for Wireless Sensor Networks, IEEE Communications, Vol. 41, Issue 2, pp. 116-125, Feb. 2003.
4. W. Heinzelman, A. Chandrakasan and H. Balakrishnan. An Application-Specific Protocol Architecture for Wireless Microsensor Networks, IEEE Transactions on Wireless Communications, Vol. 1, No. 4, October 2002.
5. V. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar and N. Shroff, A Minimum Cost Heterogeneous Sensor Network with a lifetime Constraint, IEEE Transactions on Mobile Computing, Vol.4, No.1, January 2005
6. G.W. Ng and K.H. Ng, Sensor management – what, why and how, Information Fusion, Vol. 1, Issue 2, pp. 67-75, December 2000.
7. Yang Yu, B. Krishnamachari and V. K. Prasanna, Issues in Designing Middleware for Wireless Sensor Networks, IEEE Network, Vol. 18, Issue 1, Jan/Feb 2004.
8. Chien-Chung Shen, Srisathapornphat, C., Jaikaeo, C., An adaptive management architecture for ad hoc networks, IEEE Communications Magazine, Vol. 41, Issue 2, Feb. 2003.

OPC-based Architecture of Embedded Web Server

Zhiping Jia[1], and Xin Li[2]

[1] School of Computer Science and Technology, Shandong University,
250061 Shandong, China
zhipingj@sdu.edu.cn
[2] Intelligence Engineering Lab, Institute of Software, Chinese Academy of Sciences,
100080 Beijing, China
lixin@mail.sdu.edu.cn

Abstract. This paper combines OPC (OLE for Process Control) standard to EWS (Embedded Web Server) in order to distribute an integrated service for accessing real-time and history data from control networks. A layer of OPC XML-DA (Data Access) service is added between web services interfaces and embedded operation system in OPC-EWS. Architecture of DCS (Distributed Control System) with OPC-EWS is described too. This OPC-EWS applies such technologies as OPC, Java and XML and links to Internet through TCP/IP. Therefore, accesses to all kinds of embedded devices and data share in the heterogeneous Internet/Intranet environment are implemented. At last real-time performance and security of OPC-EWS are analyzed. The experiments and application have demonstrated that the architecture has good performance and is feasible for real-time operation in DCS.

1 Introduction

As the technology of electronic measure and network communication has made great progress, DCS based on embedded-Multiprocessor has been used widely in industrial control. Whereas, embedded equipments for acquisition and control have wide variety of kinds, while interface protocols used in equipments become more sophisticated. Therefore it is highly necessary that a uniform interface be provided between every automation system vendors. OPC (OLE for Process Control) is a standard mechanism set down by OPC Foundation, making the connection of control networks with data networks in a seamless and standard mode. OPC specifications are a series of software interface protocols based upon Microsoft's COM/DCOM technology [1]. Because of OPC's advantages in the field of language independence, code reuse and easy integration, OPC has been used more and more frequently.

The thesis subscribes an OPC-EWS and DCS with this new EWS, making the connection of control networks with data networks in a seamless and standard mode. The OPC-EWS can issue static and dynamic web pages as well as provide interfaces for clients to monitor and provide devices used in control field.

The entire system has pliable and heterogeneous constitutes and flexible composite-composed configuration. It can provide real-time monitoring through Internet/Intranet for clients to access data from devices/controllers. Therefore

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 362-367, 2005.

seamless integration of heterogeneous system and data share are enforced in this way. The virtues of this architecture are listed below.

1) It can simplify the hierarchy of hardware and software in the entire plant, and reduce intermediate lays and lower the cost.

2) In comparison with Ethernet and field bus, it has broader net bands to improve the performance of communication.

3) Because it is based on TCP/IP, this architecture is very flexible, open and easy to connect to WAN.

This Architecture increases communicative speeds of front-device, changes the structure of traditional integrated real-time control system, extends the scope of control system, and enlarges the contents of available information, thus pursues the tide of Internet [2].

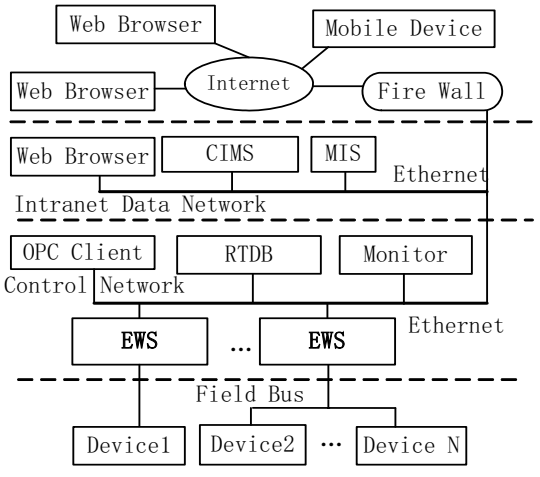


Fig. 1. Architecture of DCS based on OPC-EWS

2 Design of OPC-EWS Architecture

2.1 Structure of OPC-EWS

An average web server runs in multi-tasks (multi-process or/and multi-threads) preemptive system and it gives a firm support for every HTTP version. However, there are finite resources and it has only on web process under the Embedded OS. For this reason, Web process runs on the lowest priority level to yield to other tasks' execution [3]. Therefore, there are many differences between EWS software system and a common server.

Located between clients and EOS, EWS should provide some mechanism to monitor and control field devices, sending web pages to clients and dealing with clients' submission. It also should provide security interface for remote clients (Fig.2 shows its inter structure). The lays and modules shown in the figure tell us how the

embedded server is designed clearly. Simultaneity, the independence and transplantation of all parts can be enforced largely.

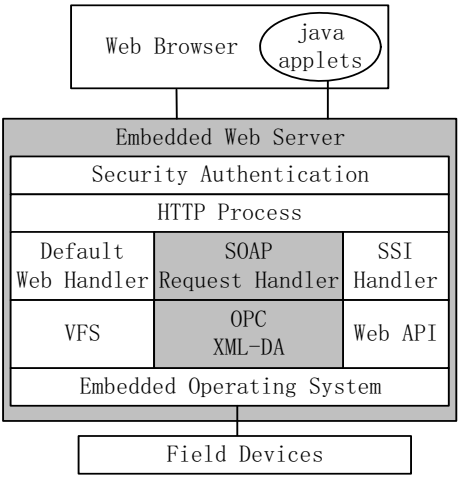


Fig. 2. Structure of OPC-EWS

A module in OPC-EWS is set to respond clients' SOAP requests. This module parses the SOAP package received, handles the contained request, accesses the corresponding real-time or historical data and formats them as an XML document. Finally, it sends them back to the client as a response. In terms of the SOAP specification, method calls and input parameters need to be XML-encoded and packed in a SOAP envelope. The manipulation of the XML documents, such as reading, formatting, parsing, construction operations, is performed using the appropriate objects' methods.

2.2 OPC XML-DA Module

OPC XML-DA module in OPC-EWS faces to field hardware in device layer and is in charge of communication to different field bus(e.g. FF, HART, Profibus) and encapsulation of manipulation for different devices and provides a standard interface for clients to access devices transparently. There are three parts in this module(Fig.3), server management, address space management and data management, which accesses device data through an inner scheduler queue. The data management part manages a data buffer. Through data-acquiring I/O composites, it refreshes value and timestamp of items and checks availability of data. Different I/O composites have the uniform interface, so when facing with different communicate protocols, e.g. Modbus and Profibus, we need only replace it with a corresponding I/O composite and other parts needn't be changed.

First of all, this module initials field bus information, gets all parameter and listens to request. When a request comes, it creates an OPC server and responses to all kinds of requests such as browser address space, add/delete group, add/delete item,

read/write, data subscription, data fresh, and so on. Polled-pull model is adopted between client and server [4]. Client sends a subscription to a certain items. When sever receives a request, it doesn't response immediately. After receiving client's Subscription- Polled-Refresh, sever gives those changed items to client.

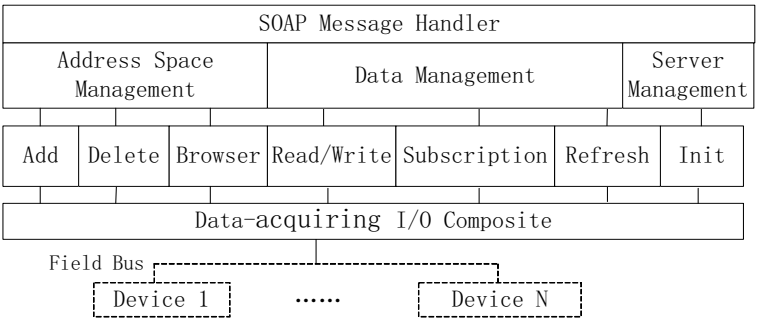


Fig. 3. Configuration of OPC XML-DA module

2.3 EWS-HTTP Protocol

HTTP is based on Client/Server and request/response model. Handling HTTP protocols is another task in OPC-EWS, which receives clients' HTTP requests, deals with them and then returns result to clients. As one running process, Web server must adopt Non-blocking I/O access model to serve for several clients. If web server took nonstop polling to requests, it would occupy most of processor's resources. It is fortunate for us that we can call 'select' method in Linux to solve this problem. The 'select' method is used to monitor client's connect requests and may ensure other clients' read/write when one client is blocked for read/write. To ensure server can deal with several clients' requests at the same time, structure variable is used in order to save all information of one connection, such as IP, browser type, protocol version, URL, head info, timeout and so on. At the same time, every connection corresponds to a life span.

OPC-EWS listens to new connections, holds old connections and handles their requests. First of all, it initials system configuration and registers, then runs as a process and listens to new connections. When a new connection request comes and the count of current connections is less than the max allowed number, it will be accepted, otherwise, refused.

OPC-EWS parses new requests: it checks request line and then saves URL and head info to the related structure. With the relation to URL, it finds request resources and checks if protection and authentication are needed. If requests come from form submission, OPC-EWS calls related CGI function; if they request web pages or other files, it reads them into buffer and checks SSI (Server Side Include), return results to clients' browser. At last, it decides whether to close connection or not in agreement with protocols and client's state.

2.4 Publishing of Real-Time Data

Today, the key problem of DCS on web is how to implement visual show of measured object (e.g. real-time curves) and refreshing real-time data. We put forward a feasible and high efficient solution — real-time monitoring combined XML with Java.

A web browser to access OPC-EWS is used by thin-client in this system. The thin–clients get all monitoring views, curves and reports in industrial field. In order to separate views from real-time data, the user should save the monitoring view in XML file. Static objects (e.g. text, label, picture…) are transmitted only once and dynamic object (e.g. LED, switch, liquid level…) are shown for client as Java applet. Java applet refreshes when receiving respondent soap message. So every client authorized can monitor all filed conditions through a web browser.

3 Discussion

3.1 Real-Time Performance of OPC-EWS

It is a vital specialty for real-time performance in embedded control system. It is also very difficult for real-time performance to be used in control system on web. To our disappointment, there is not a mature and perfect solution until the present time. In order to improve real-time performance in the whole system, three steps are implemented.

First, in the design of software, optimum program and arithmetic are used, such as real-time task scheduling queue in data-acquiring I/O composites and reducing time delay in software system.

Secondly, “subscription-refresh” model is adopted in client. Server need only send changed data. It decreases net flow enormously and enhances communication performance on network.

Thirdly, in network architecture, 100M-Ethernet and Gigabit-Ethernet become gradually common in the LAN and WAN. Net band resources are increasing fast because of the development of switch technology. These lay the foundation for the solution in hardware.

Furthermore, some attributions (e.g. RequestSamplingRate, EnableBuffering, DeadBand) in client’s request could be set to optimize performance of server to meet the expectations of real-time for data transmission, refreshing views and sending command.

3.2 Security of OPC-EWS

Security is very important in control system connected to Internet. Two common authentication algorithms, BAA (Basic Access Authentication) and DAA (Digest Access Authentication), are ready for EWS. In DAA, Password is encrypted by MD5

message-digest algorithm. So, DAA is safer than BAA [5]. There are code sources in RFC1321. MD5 Message-Digest Algorithm is granted by RSA Data Security, Inc.

There are two files used to save user name, password and visit rights in a protected directory. When a browser visiting web page, sever checks if the wanted page file is protected. If the page file needs DAA, server returns state code 401 and head of www-Authenticate, including encrypted data, realm and nonce. After Browser receives response, the client inputs user name and password. Next, browser sends them to the server to be checked. Server compares them with those from the two files. If they are the same, server returns head info of www-Authenticate (including new nonce for user to visit other protected web pages) and requested web pages.

Some APIs (e.g. add/delete user's right and add/delete files) are supplied for user management in EWS.

4 Conclusion

This paper has proposed the OPC-EWS architecture applied such technologies as OPC, Java and XML and links to Internet through TCP/IP. The DCS with OPC-EWS has pliable and heterogeneous constitutes and flexible composite-composed configuration. It can be connected to all kinds of embedded devices and supplies real-time remote monitoring on Internet/Intranet. Then seamless integration of heterogeneous system and data share are implemented. As OPC-EWS has its characteristics of seamless and standard way, it will be used widely in the future.

References

1. OPC Common Definitions and Interface Version 1.0. OPC Foundation (1998)
2. McCombie: Embedded Web server now and in the future. RealTime Magazine, No.1, March,(1998) 82-83
3. Mi-Joung Choi.an: Efficient Embedded Web Server for Web-based Network Element Management. IEEE Internet Computer (2000)
4. OPC XML-DA Specification 1.0. OPC Foundation (2003)
5. HTTP Authentication: Basic and Digest Access Authentication. http://www.ietf.org/rfc/rfc2617.txt (1999)

Synchronized Data Gathering in Real-Time Embedded Fiber Sensor Network

Yanfei Qiu, Fangmin Li, and Ligong Xue

School of Information and Engineering
Wuhan University of Technology, Wuhan, China
yanfeiqiu@tom.com, lfm68@sina.com, lgxue@yahoo.com

Abstract. An *Embedded Fiber Sensor Network* (EFSN) is a health monitoring network for large structure consisted of several sensor nodes, each equipped with fiber-optic sensor, embedded processor, and data transceiver. The collective sensing data of such a system of networked sensors can be utilized to evaluate structural health conditions of bridges, elevators, composites, mines, and even reactors. Key concerns for designing such networks are real-time detection, data synchronization and reliable information submission. This paper addresses these issues and presents an *Independent Circular Cache Queue* (ICCQ) to reduce the amount of traffic in the network. In the ICCQ mechanism, a self-managed scheme is implemented in the network to maintain stability under varying environmental disturbance and sensor sensitivities.

1 Introduction

Advances in fiber-optic sensors have enabled the measurement of deformation and temperature inside or at the surface of structures of engineering facilities, such as bridges, buildings, airplane wings, frames, vehicles and so on [1]. To monitor the whole health condition of a structure, we deploy a proper number of fiber-bragg-grating sensors to constitute a sensor network. Each sensor will automatically record the data about the degradation and aging of the structure at programmed interval, reduce and store the obtained values, and submit processed data to the upper-layer managing system, usually a mobile computer [2]. Expert diagnosis will be extrapolated through analysis of the gathered data.

Since structure diagnosis is involved with overview of the structure and only activated when queried, applications implemented in such a sensor network encounter some rigorous restrains. First, the detection of each sensor node must be real-time, or the conclusion based on its data can not instantly trace the variation of the structure. Second, each sensor should operate synchronously, that is the engaged sensor nodes detect, process and submit information within tolerable latency. Third, a stable intercommunication between sensor nodes and managing system is indispensable to access the network. We introduce an ICCQ mechanism to meet both real-time quality and synchronization. In addition, to implement TCP/IP stack in the sensor node, we utilize an embedded processor based on ARM architecture and a micro operation system called UCOS-II. Thereby, sensor data is sent using the best-effort transport

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 368-373, 2005.

protocol UDP, and administrative instructions are conducted through the reliable byte-stream transport protocol TCP.

The paper is organized as follows. In Section 2, we describe the system architecture of our Real-time Embedded Fiber Sensor Network (REFSN). Section 3 details the ICCQ mechanism. Section 4 discusses synchronization under ICCQ and intercommunication for data gathering. Section 5 demonstrates the conclusion and outlook of the REFSN system.

2 The REFSN System Architecture

A fiber-bragg-grating (FBG) sensor is a fiber photo-imprinted with Bragg Grating, which is a very selective spatial reflector. When a FBG sensor attached to a structure, the deformation or temperature of the structure will cause a proportional shift in the reflected and transmitted spectrum. By analysing the varying spectrum, the data values corresponding to the physical factors can be extracted [3] [4]. Such values will be sampled and processed in an embedded processor periodically. The sample interval $T_i, i \in (1,2,...,n)$ should be as small as possible in order to figure out the varying curve of deformation or temperature. Considering that data process will take up some time $T_{process}$ and data sending spends some time t_{send}, the embedded processor equipped in a sensor node should respond and compute fast enough. We choose a 32-bit ARM-based processor S3C44B0x to charge this procedure, which is embedded with a Fast Interrupt Query [5] (Fig. 1).

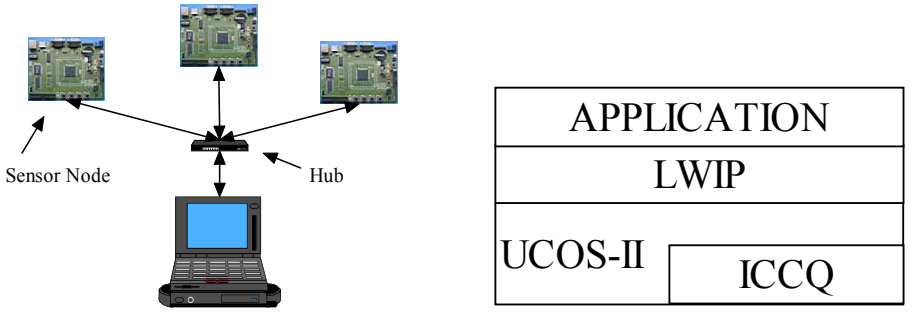


Fig. 1. The left is the EFSN System and the right is the Sensor Node Software System

Software running on REFSN involves UCOS-II [6], LWIP [7] stack and application tasks. The layer structure is shown in Fig.1. UCOS-II is an embedded operation system providing preemptive task schedule based on its priority, and LWIP is a light TCP/IP stack allowing embedded system to communicate with intranet. The main objectives of ICCQ are to: (1) sample and record the data of the FBG sensor; (2) submit data to applications for future transferring; (3) execute the practical operation while synchronizing with other counterparts.

3 The ICCQ Mechanism

Many real-time operation systems, including UCOS-II, have critical kernel section, e.g. task schedule and stack adjustment, which can not be interrupted. Such a situation also exits in LIWP stack, and will bring about random latency to reduce accuracy in real-time applications. Therefore, applications designed with embedded operation system need to consider an alternative management mechanism for real-time data gathering. In REFSN system, we utilize ICCQ mechanism to cooperate with UCOS-II and LWIP. It is consisted of three basic parts as following (Fig.2):

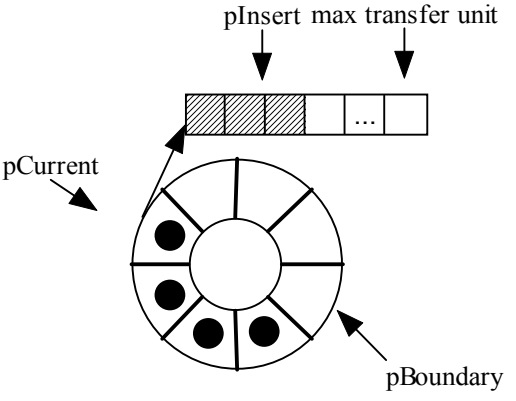


Fig. 2. Cache Queue

3.1 Circular Cache Queue

Circular Cache Queue is a combination of sequential linked lists and circular linked lists. To relieve the traffic burden in REFSN, the size of each sequential list is assigned as maximum transfer unit S_{mtu} to avoid packet fragment, and data submission occurs every T_0 time. We let S_{items} denote the maximum number of items in the cache queue. Considering memory limitation in embedded system, S_{items} should be as small as possible. Let S_{record} denotes the size of a single record, when

$$S_{mtu} > \frac{T_0 + t_{process} + t_{send}}{T_i} S_{record} \qquad i \in (1,2,...,n) \tag{1}$$

S_{items} can be evaluated as an arbitrary integral number.

3.2 Self-Managed Scheme

The FISR is an independent fast interrupt service route, not managed by UCOS-II (Fig.1). When FBG sensor completes a conversion, it sends an interrupt request signal to inform the embedded processor (Fig.3).

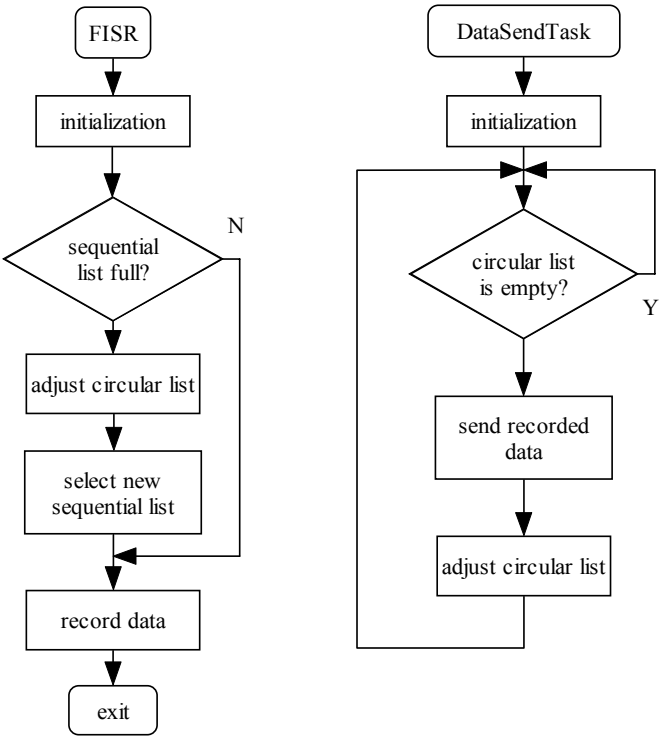


Fig. 3. Fast Interrupt Service Route and Data Send Task

3.3 Data Send Task

Data Send Task is responsible for transferring the structure information to applications and ultimately to managing system (Fig.3). This task is application task in UCOS-II operation system, and managed by the latter. Task schedule only affects $t_{process}$ and t_{send}, without reducing the accuracy of sampled data.

4 Synchronization and Intercommunication

The managing system broadcasts data gathering command amongst REFSN to query structure information. Each sensor node receiving the command sends a confirm

signal back to the managing system, and then begins to submit valid processed data. Due to LWIP, an embedded TCP/IP stack, this handshake takes a few microseconds to establish a synchronized connection. As the hardware and software in each sensor node are not exactly the same, discordance will aggregated to cause asynchronous. Thus, the handshake should be activated at a proper interval, denoted by T_{sync}. Specify $T_{max} = \max\{T_1, T_2 \ldots T_n\}$, $T_{min} = \min\{T_1, T_2 \ldots T_n\}$, then

$$T_{sync} < \frac{T_{\max} T_{\min}}{T_{\max} - T_{\min}} . \tag{2}$$

To be self-managed, environment disturbance should be taken. Since some sensor nodes may disconnect and retreat from the established network, and occasionally, a new sensor node may join into the network, thus it is time to handshake again to form a new synchronized sensor network (Fig.4).

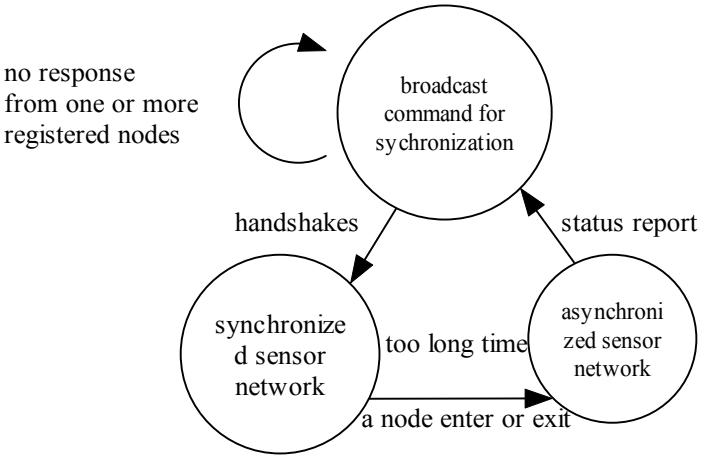


Fig. 4. Synchronization of sensor nodes

5 Conclusions

The managing system and REFSN are connected as a bus topology, illustrated in Fig. 1. Typical values of parameters are illustrated in Fig.5, as well as the human-machine interface. Press “Connect” button to send a query command, and start to gather data from sensor network. A red curve indicates the variation of the temperature at that time. Each sensor node is identified by an IP address. To display another curve amongst the sensor network, it is only needed to change the Node ID. Gauge step and scale can be adjusted in the pop-up dialog when “configure” is pressed.

Our experiment results show that the data precision is ensured and improved. In addition, the variation of the temperature of different points on a structured surface

can be reflected by the red curve, and the connected sensor network can work synchronously and stably.

parameters	typical value	unit
T_i	20	ms
T_0	1	s
$t_{process}$	0.1	ms
t_{send}	0.1	ms
S_{record}	20	byte
S_{mtu}	1024	byte
T_{max}	20.3779	ms
T_{min}	20.3763	ms
T_{sync}	250	s

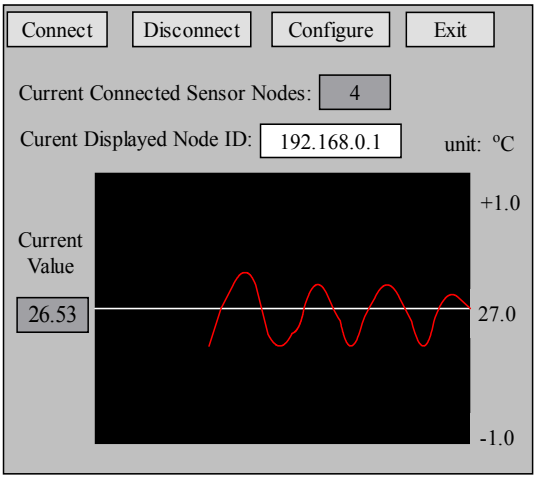


Fig. 5. Experiment Parameters and Manage Interface

In future work, the ICCQ mechanism will be implemented by Service On Chip method. It is expected to result a faster and more stable system.

Acknowledgements

This work is partially supported by the Key National Science Foundation of China (No. 50335020) and the Doctoral Science Foundation of China (No. 20020497006).

References

1. Daniele Inaudi: *Long-Gage Fiber-Optic Sensors for Structural Monitoring*, Photomechanics, Pramod K. Rastogi, vol. 77/ 2000, p. 273, June 2003
2. Adam Dunkels, Juan Alonso, Thiemo Voigt, Hartmut Ritter, Jochen Schiller: *Connecting Wireless Sensornets with TCP/IP Networks*, Wired/Wireless Internet Communications, Peter Langendoerfer, Mingyan Liu, lbrahim Matta, et al., vol. 2957 / 2004, pp. 143 – 152, January 2004
3. http://www.aip.org/tip/INPHFA/vol-9/iss-3/p24.html
4. http://www.crc.ca/en/html/crc/home/tech_transfer/bragg
5. http://www.samsung.com
6. http://www.ucos-ii.com
7. http://www.sics.se/~adam/lwip

The Energy Cost Model of Clustering Wireless Sensor Network Architecture *

Yanjun Zhang[1], Xiaoyun Teng[2], Hongyi Yu[3], and Hanying Hu[4]

Dept. of Communication Engineering Zhengzhou Information Science and Engineering Institute, (450002) Zhengzhou, P.R. China
[1]meiliguodu@126.com, [2]tengxiaoyun@sohu.com,
[3]maxyucn@sohu.com, [4]huhanying@vip.sina.com

Abstract. The advent of technology has facilitated the development of small, low power devices that combine programmable general purpose computing with multiple sensing and wireless communication capability. This article discusses the characteristics of sensor network and compares its communication mode with MANET, and then it introduces the clustering sensor network architecture. The nodes in this architecture can be classified as the following four types: cluster head nodes, active nodes inside cluster, the dormancy nodes and free nodes. An energy cost model in the sensor network is given in the article. The energy cost simulation is done and the influence of clustering on the energy cost is analyzed.

1 Introduction

Recent developments in integrated circuit (IC) technology has allowed the construction of low-cost small sensor nodes with multiple sensing and wireless communication capabilities that can form distributed wireless sensor network systems. These systems can be used to perform detection, localization, tracking, and identification of objects in diverse military, industrial, scientific, office, and home applications. A large amount of low-cost intelligent microsensors can be rapidly deployed in an environment of interest. These sensors can individually sense the environment. They can also collaborate with each other and achieve complex information gathering and dissemination tasks.

Generally speaking, in communication mode senor network is quite similar to the MANET. They are both adoption infrastructure, multi-hop wireless communication manner, and each node of the two networks has the ability to forward the packet. But in detail there are many differences between them [1], mainly including the points as the following.

Sensor network is built to obtain the information availably, and communication in sensor network is just an assistant means for that purpose. But the MANET is a sheer communication network. In sensor network, the data sent from the detect node to the

* This work is supported by the National Natural Science Foundation of China (Approved No.60472064).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 374-380, 2005.

sink node may be aggregated to reduce the data to deliver. But the data in MANET can't be modified when transmitted from the source node to the destination node. The scale of sensor network is larger than that of MANET. The number of the nodes in sensor network is always many decuple with in MANET. In sensor network, the density of nodes is very high, and sometimes it can amount to 20 in a stere[2]. The nodes in sensor network often expire for the using up of the battery or other reasons. The change of network structure in sensor network is often due to the node breakdown but in MANET it is often due to the change of the node position. The adoption of the communication mode in the network is mainly broadcast or multicast, but that in MANET is often P2P.

On the other hand, senor network has some characteristics is different with other types network. There are large numbers of nodes in sensor network, thus the management of network is difficult. The battery of the node is limited and unchangeable; therefore, the network designers must take the energy control into account in the first place. The cost of the node is cheap but the stability of the node is poor. The fault tolerant mechanism is important in sensor network design. Sensor network synthesizes sensor technique, embedded computing technique, distributed information processing technology and wireless communication technique. Sensor network can be applied in many domains, such as national defense military, national security, environment monitor, transportation management, medical treatment hygiene, manufacturing industry, anti-disaster, etc. We can say that the appearance of sensor network is a revolution in the realm of information collection.

Owing to the above reason, the network structure and management protocol of MANET can't be applied to sensor network.

2 The Architecture Model of Sensor Network

To facilitate scalable operations within sensor networks, sensor nodes should be aggregated to form clusters based on their power levels and proximity.

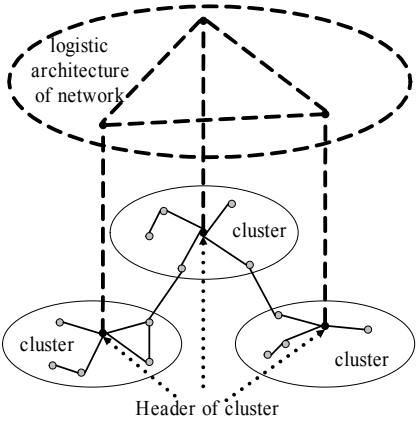


Fig. 1. Logistic architecture of sensor network

The aggregation process could also be recursively applied to form a hierarchy of clusters. Within a cluster, a cluster head will be elected to perform information filtering, fusion, and aggregation, such as periodic calculation of the average temperature of the cluster coverage area. In addition, the clustering process should be reinitiated in case the cluster head fails or runs low in battery power. In situations where a hierarchy of clusters is not applicable, the system of sensor nodes is perceived by applications as a one-level clustering structure, where each node is a cluster head by itself. Head node manages other nodes in cluster and charges the data fusion. It can reduce the pressure of network management when the network scale increases. If using the hierarchical clustering, sensor network can be regarded as a network constituted of many sensor clusters [3].

Sensor network is limited in the node ability. If the network designer adopts hierarchical clustering, the head cluster node at higher layer can not be competent. In application, it shouldn't be bigger than two classes. To save the energy of the sensor nodes, we must think over the problem of energy cost. Saving energy may be considered from the application layer, the network layer and MAC layer. In this article, we mainly consider the influence of the network architecture.

2.1 The Node Status and Function

There are four types of nodes in clustering sensor network, namely, cluster head node, active node inside cluster, the dormancy node and free node. The type of each node is decided by its place in the network. Then the function of each type is introduced in a specific way.

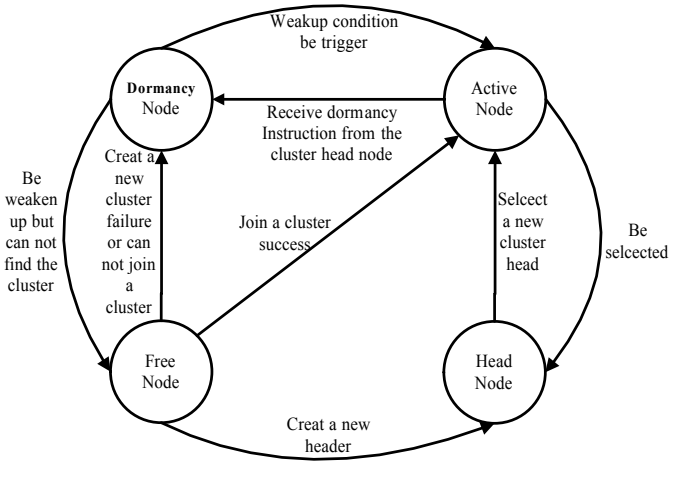


Fig. 2. Nodes types' conversion

The nodes with more surplus energy would be selected to be cluster head nodes according to the principle of the opposite position stability. Cluster head node is responsible for sending the data to carry on the fusion and the consociation processing

to the node inside the cluster, according to the fact whether data suit requests the claim valuation current or not. It maintains the local routing information in the cluster and the routing information with its neighbor clusters. When the topology changes, it will renew the local routing information. According to the correlation of the node inside the cluster, cluster head node decides dormancy node and notifies it to turn into the dormancy. According to the request of the system, cluster head node judges whether the data in the current node memory satisfy the request or not. When the cluster head node is aware of the fact that it is no longer competent for the work, it will start an election, and deliver the native information of maintenance to new head node, and establish itself as an active node inside cluster. The new head renews the native information toward the member inside the cluster after receiving the information. If a head node is suddenly expired, its neighbor's nodes will start an election.

The mission of data collection is completed by the active node inside cluster. The active node periodically or aperiodically renews the information toward the head node. The active node reports the new position information to the head node when the position information has changed; it carries on the backup information of the head node, and monitors the suddenly expiring of head nodes. If the expiring occurs, it will broadcast the expiring message toward the nodes inside the cluster, then start an election, and deliver the backup information to new head node.

When active node inside cluster receives dormancy instruction from the cluster head node, it will become the dormancy node. The node will set the trigger conditions to wakeup itself before entering the dormancy. Once wakeup condition is triggered, the node will be awaken.

The node which doesn't exist in any cluster is called free node. When a node becomes a free node, it will start a new cluster or join a cluster to get away from the free status. If free node fails to start a cluster or can't join a cluster, it will turn into dormancy node.

2.2 The Energy Cost Model of Clustering Sensor Network

According to the analysis of the energy cost, we built up the following model. The energy cost includes two parts, communication cost and MCU processing cost. The management cost is also an important issue in clustering sensor network. Management cost is composed of the computing of data fusion cost and management communication cost. We set the following assumptions before the simulation of the sensor network energy cost.

The total node number in the network is N, and the number of sampling nodes is n. Sampling frequency is f. The length of data packet is m1, and the length does not change after the data fusion. The energy cost of sampling in unit time is c1; data processing is c2.The length of management packet is m2 and the frequency is fm in the cluster. The communication mode is P2P when the active nodes in the cluster send packet to head node, otherwise the mode is multicast.

We can get the following expressions. If using the flat Architecture, the energy cost in unit time is W_s .

$$W_s=n\times\left(c1+f\times H\times x\times m1\times\left(Tx+Rx\right)\right)+N\times c2 \tag{1}$$

The x is a random disturbing. The cost of sending a bit data is Tx, and that of receiving is Rx.

$$W_c = p1\times\left(\sum_{i=1}^{h}\left(\frac{8i}{s}\times\left(p2\times(Rx+Tx)\times(c1+i\times f\times m1)+f\times\left(H+(-1)^i\times i\right)\times x\times m1\right)\right)\right) \quad (2)$$
$$+N\times c2+W_m$$

W_c is the energy cost in unit time in cluster Architecture.

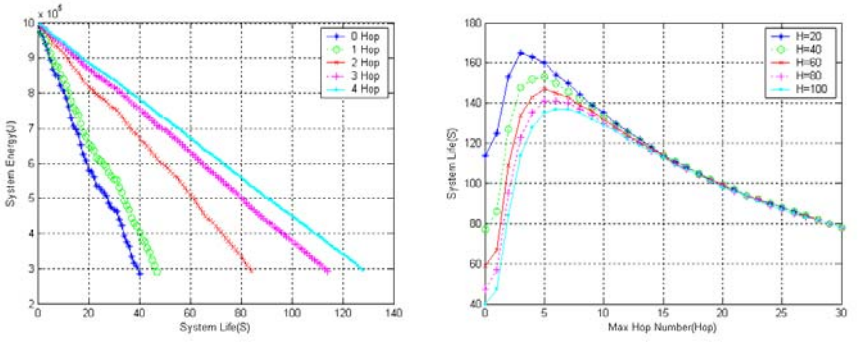
W_m is the cost of management communication.

$$W_m = f_m\times p1\times\left(\left(8\sum_{i=1}^{h}i^2+1+4\sum_{i=1}^{h}\left(\left\lfloor\frac{1+2i}{3}\right\rfloor-1\right)\right)\times(Rx+Tx)+8h\times Rx\right) \quad (3)$$

The total number of clusters in the network is P1, and the average number of the sampling nodes in a cluster is p2.

3 Simulation and Analysis

Our simulated network consists of 99255 stationary sensor nodes distributed in a grid pattern. m1 is set to 64*8bit,and m2 is equal to m1. c1 is set to 3*0.002J and c2 is 3*0.0104J. To analyze the energy cost of the sensor network, we get simulation result figures shown below.



(A) (B)

Fig. 3. Simulation results(1)

We can see from Figure 3-A that the network life will extend with the increase of the nodes number in a cluster. It means that the energy cost can be reduced by adopting cluster in sensor network. In Figure 3-B, when the network life reaches its peak value, network life will decrease along with the growth of the nodes number in a cluster.

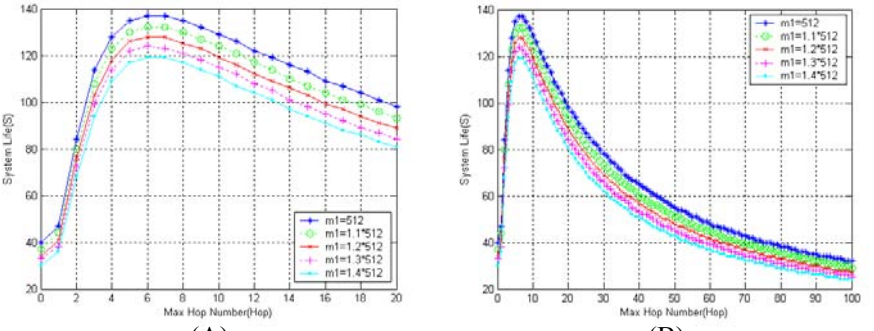


Fig. 4. Simulation results(2)

Figure 4-A indicates the peak value of the network life appears at the same cluster scale when we change the length of data packet. It means that the length of data packet has no influence on the relationship between the nodes number and the network life. It only reduces the energy cost of the data transmission. When the number of hop is increased, we can see it clearer from Figure 4-B.

We get the conclusion from Figure 5-A that if the ratio of the sampling nodes number to all nodes number in the network is less than 10 percent, the adaptation of the miniature cluster can not reduce the energy cost, on the contrary, the cost will increase because of the cost of management in the cluster. Figure 5-B is same to Figure 5-A except that the maximum of hop number is one hundred. From Figure 5-B we can see that when the hop number is bigger than twenty, the system life is decrease with the hop number increasing.

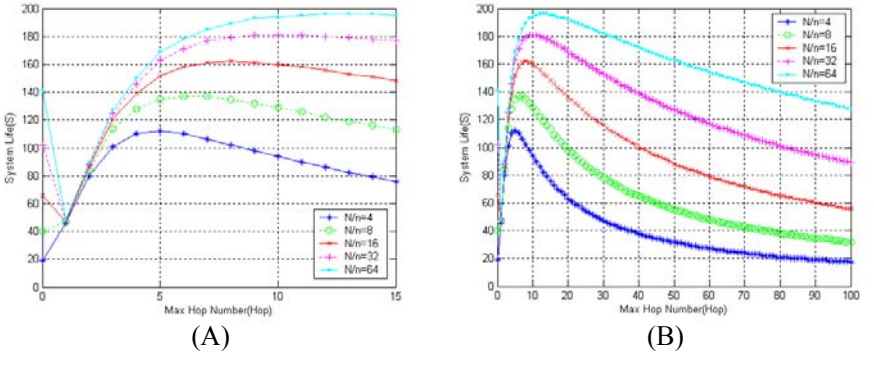


Fig. 5. Simulation results(3)

From above analysis we can draw the following conclusion，the energy cost has certain relationship with management cost and the scale of the cluster in the network, the number of sampling nodes. Clustering perhaps increases the energy cost for the management cost when the operation data flow is fairly small.

4 Summary and Future Works

In this article, clustering sensor network architecture has been introduced, and a sensor network energy cost model is given. The simulation of energy cost has been done and we have analyzed the factors related to the energy cost.

In order to optimize the clustering sensor network architecture and reduce more energy cost, the influence of dormancy mechanism on sensor network life needs further investigation. The investigation of the energy cost model which we have constructed will be further continued.

References

1. Perkins, C., “Ad Hoc Networks, Addison-Wesley”, Reading, MA, 2000.
2. Shih, Cho, S., Ickes, N., Min, R., Sinha, A., Wang, A., Chandrakasan, A., “Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks,” Proceedings of ACM MobiCom’01, Rome, Italy, pp. 272-286, July 2001.
3. Heinzelman, W., “Application-Specific Protocol Architectures for Wireless Networks,” Ph.D. thesis, Massachusetts Institute of Technology, 2000.
4. Tian D. and Georganas, N. D., “A coverage-preserving node scheduling scheme for large wireless sensor networks,” in Proceedings of the first ACM international workshop on Wireless sensor networks and applications, pp. 32-41, 2002.
5. Heinzelman W.R., Kulik J., Balakrishnan H., Adaptive protocols for information dissemination in wireless sensor networks. In: Proceedings of the ACM MobiCom’99. Seattle: ACM Press, 1999. 174~185.

Traffic Control Scheme of VCNs' Gigabit Ethernet Using BP

Dae-Young Lee and Sang-Hyun Bae

Dept. of Computer Science & Statistics, Chosun University, Korea
375 Seosuk-Dong, Dong-Gu, Kwang-ju, Korea 501-759
Tel : +82-062-230-7962 Fax : +82-062-234-4326
cssna01@chosun.ac.kr

Abstract. VCNs (Virtual-Connection Networks) of gigabit-ethernet can be efficiently used to transport packet data services. The switching system will support voice and packet data services simultaneously from end to end applications. To guarantee quality of service (QoS) of the offered services, source rate to send packet data is needed to control the network overload condition. Most existing control algorithms are shown to provide the threshold-based feedback control technique. However, real-time voice calls can be dynamically connected and released during data services in the network. If the feedback control information delays, quality of the serviced voice can be degraded due to a time delay between source and destination in the high speed link. An adaptive algorithm based on the optimal least mean square error technique is presented for the predictive feedback control technique. The algorithm attempts to predict a future buffer size from weight factor (slope) adaptation of unknown functions, which are used for feedback control. Simulation results are presented, which show the effectiveness of the algorithm.

1 Introduction

Currently GEA(Gigabit Ethernet Alliance) composed of a few companies supporting Ethernet, based on the speed that could create 10 Gbps is getting ready to make Ethernet as the all-weather network technology such as introducing technology to make SONET and WDM compatible[1]. Furthermore, the best QoS can not be provided when multimedia services sensitive to delay such as voice and images are to be sent since most of the broadband wired-wireless communications network to be advanced in the future is based on IP, although IP is feasible when using the network transmitting only data traffic. In order to overcome this problem, various queuing technologies are being used at the protocol level such as RSVP(Resource reSerVation Protocol), which provides the standard method to obtain the guaranteed bandwidth in the IP network infra environment[2]-[3]. However, most studies examine only some areas of the ever-increasing existing network and broadband wired-wireless communications network that need to transmit new multimedia data. Since the network management technologies that realize performance improvement through this approach

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 381-387, 2005.

are limited in the range of application, modeling and controlling technologies are needed in new paradigm[4]-[5].

In order to resolve this problem, improved feedback control information and algorithm are proposed in the present study as one of the solutions for effective control within the timeout period after the dynamic connection has been set to monitor past system records such as input ratio and buffer size and to predict the future system status by investigating the network status. For these purposes, the standard least mean squared error, *NLMS,* algorithm is introduced to analyze and control the upper level of the communication network[6] to provide the best QoS(Quality of Service) to the users and to set the optimal model for the analysis and design of network software and hardware factors by increasing the channel utilization usage ratio. These results could be used to maximize the effective bandwidth use of large-scale communication network, to verify protocol, and to develop conversion technology. The improved feedback control information, model, and algorithm are discussed in the second section and the results of simulation done using both the *ER* feedback method and proposed *NLMS* feedback method are shown in the third section.

2 A Predictive Feedback Control Model

2.1 A Proposed Predictive Feedback Control Model

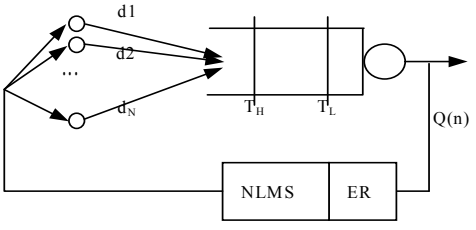


Fig.1. Feedback Control Model

A proposed predictive feedback control model is presented in Fig. 1 above. N sources transmit packet data cells in a single switch, a cell transmission rate is constant, and a queue state is monitored regularly. It is assumed that a transmission delay time of packet data between a source and a switch is di, and that sources is added or deleted randomly for ABR service. A network state is specified in time n by Q(n) of a queue length at a switch node. For a given ABR traffic processing buffer, T_H and T_L show high and low thresholds respectively. A predictive control function computes a future queue length in response to time-series by a queue length. When a future predictive queue size exceeds the high threshold T_H, the switch is considered to be in a congestion and the switch computes the explicit rate(ER) at which sources have to send a backward RM cell to the switch in order to avoid a congestion. If it is less than the high threshold T_H, however, a source changes its transmission rate in its computation of ACR (Available Cell Rate) by being informed of non-congestion situation instead of ER.

2.2 A Predictive Control Function Using NLMS

NLMS control estimates buffer size in the next k steps using a linear function with a current value of the buffer size and weighting factor(slope) at time n.
Let Q(n) denote the buffer size at time n. The k-step predictor is formulated such that the buffer size at k steps in the future is estimated from the Q(n), as given by

$$Q(n+k) = a^k(n)Q(n) \tag{1}$$

Here, *a(n)* is the weighted factor estimated at the time *n*, and *k*=1, 2, …,*t* is the maximum predictive range. The predictive error at the time, *n* is $e(n) = Q(n) - \hat{Q}(n)$

$$\hat{Q}(n) = a(n-1) - Q(n-1) \tag{2}$$

The prediction scheme uses the error to modify the weighting factor whenever the error is available at each time step. Furthermore, the weighting factor *a(n)* is affected in time as sources are added or removed and as the activity levels of source changes. We thus put the problem into the one of estimating the weighting factor and use the normalized least mean square error (NLMS) linear prediction algorithm. Given an initial value for *a(n)=0*, the weighting factors are updated by

$$a(n) = a(n-1) + \frac{\mu e(n) Q(n-1)}{|Q(n-1|^2} \tag{3}$$

Here, μ is a constant. When *Q(n)* is the apex, *a(n)* is converged at mean square as the optimal solution. The normalized least mean squared error, *NLMS*, is not affected significantly by the factor, μ. At each time step, the weighted factor , *a(n),* indicates the direction of function changes on whether the buffer size increases or decreases according to the residual *e(n)*, which is the difference between the actual buffer size, *Q(n),* and the estimated buffer size, $\hat{Q}$.

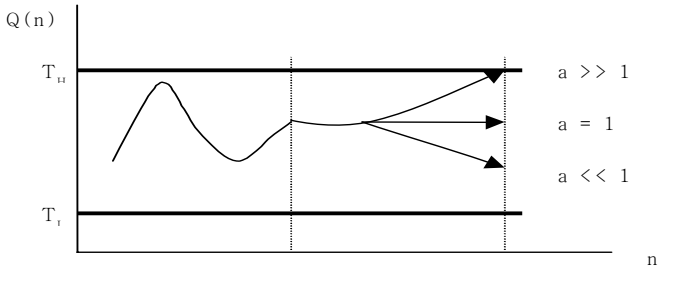


Fig. 2. The Prediction Scheme

Therefore Fig.2. is the predictive plan suggested in the present study. When *a* >>1, the predicted buffer size increases according to the equation, $Q(n+k) = a^k(n) Q(n)$. The time, *k*, agreeing at T_H, could be predicted beforehand using *Q(n)* and *a(n)* shown at the time *n*.

2.3 A Predictive Control Function of Neural Network Using BP

A non-linear predictive function using neural network adjusts to predict a optimized value using BP algorithm[2]. It computes optimized variables of a non-linear equation (sigmoid) included in neural network nodes, and adjusts to get minimal errors to be occurred in a predictive value. BP is a kind of delta learning method to adjust adaptively the degree of a connection in order to minimize the differential error between required output and predictive output. Input layer x_i gets continuously changing queue length $Q(n), Q(n-1),\ldots,Q(n-m-1)$ in time units, and output layer gets a predictive value of queue length $Q(n+k)$ after $n+k$. A case using neural network as in using NLMS also predicts a future queue length through monitoring queue length at a switch. However, the case is more complicated than the case of NLMS, because a weighted value for each connection link should be computed in advanced for optimal adaption. The detailed computation processing of BP algorithm is consulted in Reference [2].

3 Simulation

3.1 Simulation Environment

As in Fig. 3, the simulation model of a control algorithm presented in this paper is that the link speed of the switch is set to 150 Mbps, and the link speed for each source to the switch is set to 150 Mbps/N for N sources. The control algorithm is experimented in Visual C++ for a single bottleneck switch with a buffer of high and low thresholds of 5000 and 1000 cells respectively.

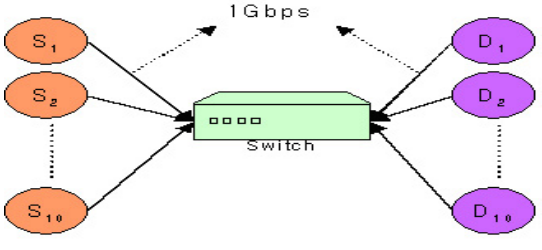


Fig.3. Simulation Model

Following parameters are used for the simulation: peak cell rate(PCR) is set to 150 Mbps, additive increase rate(AIR) is set to 0.1 Mbps, and explicit reduction factor(ERF) is defined to 4/5. Ten active sources with various packet data cell generation times are used for simulation. In order to examine the transitional behaviors, an abrupt change of active sources is made. Initially, sources with cell generation times at {2, 4, 6, 9, 11, 13, 16, 20, 21, 23} are active, in which the numbers represent the time-delay *di* from current time-unit n at the switch to the sources. At time-unit 1000, sources with time-delay *di*{14, 16, 17, 19, 20, 22, 23, 26, 28, 30} are active, in which it includes the active sources with long delays. Two cases are compared in terms of a

stabilization and a congestion of queue length at the switch through the change of transmission delay. The first case uses only feedback control method, and the second one does feedback predictive control method.

3.2 Simulation Results

Fig. 4 presents the change of queue length size at each switch, one of which uses a feedback predictive control algorithm using NLMS proposed in this paper, and the other of which uses only a feedback control one. A predictive interval(k) for NLMS was 10. Fig 4 presents that a feedback control algorithm only always brings about a congestion, and that a variation of queue length is considerably severe. It also shows that a variation of the length size Q(n) is severe after time 1000, which means that the sources with much longer delay time than other ones are incoming at the same time.

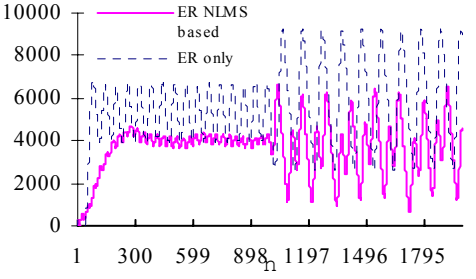


Fig. 4. Comparison of ER only and ER with NLMS

Control algorithm presents no variation of queue length before time unit 1000, close to high and low threshold, compared with a feedback control one only. After time unit 1000, however, as the sources with much longer delay time than other ones are incoming at the same time, the variation occurs more severely than before 1000 even with the predictive control method, and cases exceeding over high and low threshold also occur. The reason is that feedback delay of transmission sources is longer from time unit 1000, and that as a worst condition any traffic does not occur during time delay 1 through 13. That is, a predictive control function responds inappropriately to constant long-term interval, or sudden and random change. However, it is concluded that a predictive control algorithm causes a stability of the change and not severe congestion during simulation, compared with a feedback control one only. The use of neural network structure brings out similar results. It also responds inappropriately after n=1000.

In the switch buffer size change for the *ER* method and *ER* algorithm using *NLMS*, the buffer size before the time unit is significantly more effective compared with when using only *ER* algorithm. However, according to the influx of other sources having more time delay at the worst situation after time change as in the simulation in Fig.4., NLMS algorithm is modulated by applying BP algorithm in order to stabilize the unstable buffer size and severe changes by applying BP algorithm.

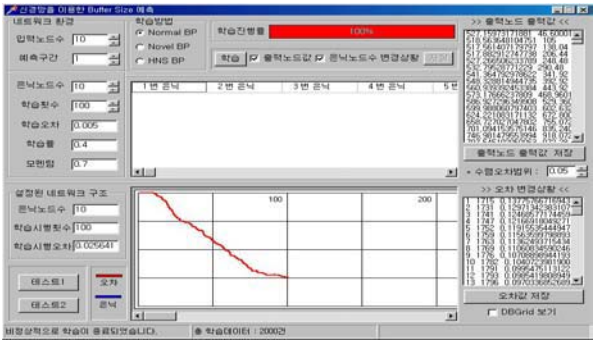


Fig. 5. The learning result on the prediction of buffer size using BP

Fig. 5. shows the results of executing the BP program to modulate the buffer size change after the time conversion point using BP algorithm, showing that the error rate graph value at the middle section of the lower section of the figure drastically decreases.

4 Conclusions

The present study is conducted to control traffic by predicting congestion before the development of congestion at the switch in gigabit Ethernet VCNs and the buffer size. The results on predicted buffer size are actively used as feedback control information so that expedite and accurate congestion status could be reported. The prediction algorithm based on the *NLMS* predictive plan is applied in the control algorithm based on *ER* algorithm using the normalized least mean squared error method, *NLMS*, by predicting the buffer size after the *k* step.

Currently, despite the continuously golden age of xDSL (x Digital Subscriber Line) technology in the high-speed Internet access market in Domestic, Internet users think that it takes too long time for website searching and content download to the point that they still call WWW (World Wide Web) as WWW (World Wait Web), showing that the network congestion control technology has solved the problem only partially. In other words, the factors such as slow access speed due to explosive traffic increase, unpredictability of sudden disconnection, and the limitation of usable bandwidth can not be analyzed clearly only with the existing modeling analysis.

The proposed least mean squared error method, the prediction algorithm using *NLMS*, could provide the earnest analytical method to resolve this problem with congestion.

The expansion of the technologies obtained based on the results of the present study in the future TCP/ IP based multimedia communications and wireless network areas could lead to the development of generalized communication network analytical method on all modes of communications. Furthermore, more effective application of the system analysis such as IVHS (Intelligent Vehicles Highway Systems) and control technology would be the steppingstone for researches for the development of application technology for the actual application on systems.

References

1. H. T. Kung, "Gigabit Local Area Networks: A Systems", IEEE Communications Magazine, Vol. 30, Issue 4, pp. 79-89, April 1992.
2. Jean Walrand and Pravin Varaiya, High-Perf ormance Communication Networks , The Morgan Kaufmann, 2000
3. William Stallings, "High-Speed Networks; TCP/IP and ATM Design Principles", Prentice Hall, 1998.
4. K.-H. Cho and J.-T. Lim, "Supervisory Rate-Based Flow Control of ATM Networks for ABR Services", IEICE Trans. on Communications, vol. E81-B, no. 6, pp. 1269-1271, 1998.
5. Seung-Hyub Lee and Kwang-Hyun Cho, "End-to-End Congestion Control of High-Speed Gigabit-Ethernet Networks based on Smiths Principle", Proc. of the ITC-CSCC 2000, pp.101-104, Pusan, Korea, July 2000.
6. S.P.Bhattacharyya, H.Chapellat, L.H.Keel, "Robust Control", Prentice Hall, 1995

A Jitter-Free Kernel for Hard Real-Time Systems

Christo Angelov and Jesper Berthing

Mads Clausen Institute for Product Innovation, University of Southern Denmark
Grundtvigs Alle 150, 6400 Soenderborg, Denmark
{angelov, berthing}@mci.sdu.dk

Abstract. The paper presents advanced task management techniques featuring Boolean vectors and bitwise vector operations on kernel data structures in the context of the *HARTEX*$_{TM}$ hard real-time kernel. These techniques have been consistently applied to all aspects of task management and interaction. Hence, the execution time of system functions no longer depends on the number of tasks involved, resulting in predictable, jitter-free kernel operation. This approach has been further extended to time management resulting in a new type of kernel component, which can be used to implement timed multitasking - a novel technique providing for jitter-free execution of hard real-time tasks.

1 Introduction

Modern embedded systems have to satisfy stringent requirements with respect to system safety and predictability. Currently, there are two approaches for engineering predictable embedded systems: static scheduling vs. predictable dynamic scheduling. The former approach is widely used with safety-critical systems. However, static scheduling has a major disadvantage: its use results in closed, non-reusable systems that are difficult to reconfigure and maintain. This is in contradiction to the requirement for an open system architecture that has to provide support for software reuse as well as in-site and on-line reconfiguration.

The latter approach is more promising but it requires the development of a new generation of *safe* real-time kernels, which provide a secure and predictable environment for application tasks through predictable task scheduling and interaction, extensive timing and monitoring facilities, and last but not least - predictable behaviour of the kernel itself. Such functionality cannot be efficiently accomplished using conventional kernel algorithms and data structures, i.e. linked lists used to implement system queues. Extensive linked list processing introduces substantial and largely varying overhead known as *kernel jitter* [4].

This paper presents a novel kernel design using Boolean vectors (bit-strings) in order to implement system queues. Consequently, queues can be processed through bitwise Boolean operations resulting in jitter-free operation and substantial reduction of kernel overhead. That approach has been consistently applied to all aspects of task management and interaction: task scheduling, task execution request generation, task synchronization and communication [2, 3]. It has been further extended to time management resulting in a new type of kernel component - the Static Time Manager,

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 388-394, 2005.

which can be used to implement *Timed Multitasking* - a novel computational model providing for jitter-free execution of hard real-time tasks [5].

The paper focuses on advanced task and time management based on Boolean vector processing as implemented in $HARTEX_{TM}$ - a timed-multitasking version of the *HARTEX* kernel [2]. The paper is organized as follows: Section 2 gives an overview of the kernel architecture and functionality. Section 3 presents task management using Boolean vectors. Section 4 presents advanced time management and the so-called Static Time Manager in the context of timed multitasking. The last section contains concluding remarks highlighting the implications of the developed kernel architecture.

2 *HARTEX* Architecture

HARTEX (*HArd Real-Time Executive for Control Systems*) is a safe real-time kernel, which has been conceived as an operational environment for component-based applications conforming to the *COMDES* model of distributed computation [1, 2]. Subsequently, a number of versions have been developed, i.e. $HARTEX_{AVR}$ [3], $HARTEX\mu$ and $HARTEX_{TM}$ - a timed multitasking version, which is presented in this paper.

The *HARTEX* architecture exhibits a number of novel features, which are briefly summarized below:

- *Component-based* architecture supporting kernel reconfiguration and scalability
- Integrated task and resource scheduling via the *System Ceiling Priority Protocol* – a non-blocking synchronization protocol providing for predictable and efficient management of shared resources
- Boolean vector processing of kernel data structures, resulting in very low overhead and constant execution time of system functions, hence *jitter-free operation* of kernel subsystems
- Predictable *jitter-free execution* of real-time tasks in a distributed timed multitasking environment, using an advanced clock synchronization mechanism and a new type of time manager - the Static Time Manager
- Event notification via Boolean vector semaphores, providing for the *instantaneous* broadcast/multicast of events to multiple receiver tasks
- Integrated communication protocol supporting *transparent* content-oriented message addressing within local and/or remote interactions, including both state message and event message communication

However, the main innovation of the kernel is the use of Boolean vector processing, whereby linked-list queues have been substituted by Boolean vectors, resulting in efficient and highly deterministic (jitter-free) behaviour characterised by very low overhead and *constant* execution time of kernel operations, independent of the number of tasks involved.

The following discussion presents Boolean vector processing techniques for task and time management used in the recently developed timed-multitasking version of the *HARTEX* kernel.

3 Jitter-Free Task Management

Task management is carried out in accordance with the state transition diagram shown in Fig. 1-a. It consists of the *active* task superstate, as well as three other states - *suspended, inactive* and *disabled* task. The *active* superstate encapsulates three other states - *ready, running* and *preempted* task. It is only active tasks that are recognized by the scheduler, which is the main component of the task manager.

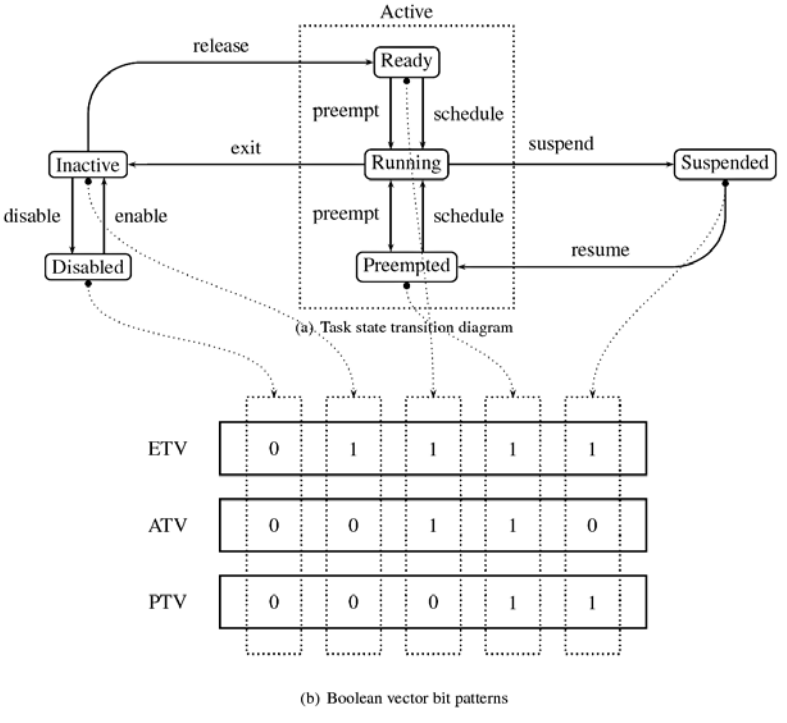


Fig. 1. *HARTEX* state transition diagram and Boolean vector bit patterns

An extended task may be temporarily suspended while waiting for some kind of event to take place, whereas basic tasks may never be suspended. A basic task will switch to an inactive state upon exiting the system, and it will be eventually released again. Finally, basic tasks may be temporarily or permanently disabled. However, this action has a run-to-completion semantics: the task is actually disabled after it has become inactive, i.e. after it has released its resources and left data structures in a consistent state. Subsequently, such a task can be released again only after it has been re-enabled.

Task management can be substantially speeded up by abandoning the traditional linked-list implementation of the dispatch queue. Instead, it can be emulated using two Boolean vectors - the *Active Tasks Vector (ATV)* and the *Preempted Tasks Vector (PTV)* that have as many bits as there are tasks in the system. Vector bit position represents task priority as well as task number, which is used as an index to a Task

Control Block (TCB) within the TCB table. Another vector of the same dimension is used in order to define the tasks that are enabled in the system, i.e. the *Enabled Tasks Vector (ETV)*.

The state encoding logic for various task states is shown in Fig. 1-b. It has been specifically designed to reduce the execution time of task state transition operations and corresponding task management primitives, which are effectively reduced to bit manipulation. Moreover, the use of the above encoding technique makes it possible to simultaneously execute multiple identical operations involving different tasks through bitwise operations on Boolean vectors. This is done in *constant* time, no matter how many tasks are involved in the operation, as illustrated by the primitive *release(tasks)*. The latter generates multiple execution requests for a subset of tasks specified by the *tasks* argument vector:

```
release(tasks)
{
    ATV = ATV OR (tasks AND ETV);
}
```

The execution time of the above function is in the (sub)microsecond range depending on the platform used, e.g. 4 μs in an 8-bit *ATmega 103* microcontroller running at 4 *MHz* and operating on 16-bit vectors. With linked-list queues, the same operation might take from tens to hundreds of microseconds depending on the number of tasks involved, even in relatively high-end processors, such as the 32-bit *Motorola 68020* [4].

Likewise, task scheduling is facilitated by the Boolean vector data structure. The task manager determines the highest priority active task to be executed by finding the highest-priority non-zero bit in the *ATV*. This is done via a bit search procedure, which takes constant time to execute, and in some processors this can be accomplished with a single instruction (e.g. the Intel BSF instruction). In fact, the scheduling algorithm is somewhat more involved because it implements integrated task and resource scheduling using the System Ceiling Priority Protocol (for more information see [3]).

4 Time Management for Jitter-Free Execution of Real-Time Tasks

This section presents static time management, which can be used to implement precisely timed transactions in the context of *Timed Multitasking* – a novel computational model combining the predictability of statically scheduled systems with the flexibility of dynamically scheduled systems [5]. This model has been recently extended for distributed embedded systems involving communication I/O drivers, as defined in the *COMDES* framework [1].

Timed system transactions are conducted by means of the so-called *Static Time Manager (STM)*. This is a dedicated kernel component that executes a cyclic static schedule with respect to *timed input/output* and *task release* actions. However, released tasks are executed in a preemptive priority-based environment provided by

the *HARTEX*$_{TM}$ Task Manager. The above schedule may be implemented as a table consisting of records corresponding to specific instants of the system (super)period.

Table records have the following format:

{ *offset, tasks_with_deadline, output_drivers, tasks_to_release, input_drivers* } ,

where each instant is specified with an *offset* from the beginning of the superperiod. Accordingly, *tasks_with_deadline* is a Boolean vector specifying the tasks whose deadline expires at the time instant given by *offset*; *output_drivers* specifies the output drivers to be executed at that instant if the corresponding task deadlines have not been violated; *tasks_to_release* is another Boolean vector specifying tasks that have to be released, and *input_drivers* specifies the input drivers to be executed at that same instant.

The Static Time Manager processes the scheduling table in a cyclical tick-driven manner that can be described with the following pseudocode:

```
with every clock tick do {
   update current time;
   if ( current_time < schedule[I].offset) return;
   else {
      generate a deadline violation vector if some
      tasks_with_deadline have not yet finished
      execution, and disable those tasks;
      execute the output_drivers of tasks that have
      finished execution;
      release tasks specified by the tasks_to_release
      vector (if not disabled);
      execute the input_drivers of released tasks;
      I = I + 1 (mod (schedule length));
      invoke the Task Manager;
   }
}
```

The above algorithm is executed using Boolean vector processing techniques, as follows:

- The deadline violation vector is generated by ANDing the *ATV* and *tasks_with_deadline* vectors. That vector is sent to the Deadline Exception Handler (if non-zero), in case some of the tasks specified by the *tasks_with_deadline* vector have not finished execution, and these are disabled by resetting their *ETV* bits.
- The output drivers specified by the corresponding vector are executed, i.e. the output drivers of tasks that have a deadline at that instant. However, a driver is executed only if the task has finished before its deadline. This can be accomplished by disabling certain drivers as indicated by the task deadline violation vector.
- Tasks specified by the *task_to_release* vector are released, i.e. registered in the *ATV* if enabled (see algorithm given in section 3).

- Input drivers specified by the corresponding vector are executed, and in particular - the input drivers of those tasks that have been actually released at that time instant.
- Finally, the task manager is invoked in order to re-schedule the processor, since some of the newly released tasks may have higher priority than the currently running task.

It is obvious from the above discussion that the presented timing mechanism implements a static schedule with respect to timed input/output and task release actions, whose operation is largely similar to that of a rotating drum sequencer used in some mechanical devices. Therefore, it has been denoted as the *Static Time Manager* (and alternatively – the *Drum Sequencer*).

With this algorithm task I/O drivers are *atomically* executed at precisely specified time instants, whereas tasks are executed in a preemptive priority-driven environment and may have termination jitter. However, jitter is effectively eliminated, as long as tasks finish execution before their deadlines.

5 Conclusion

The paper has presented advanced task and time management featuring Boolean vectors and parallel (bitwise) vector operations on kernel data structures, in the context of the timed-multitasking version of the *HARTEX* real-time kernel. The use of Boolean vectors has resulted in the elimination of linked lists, hence low overhead and jitter-free execution of kernel operations whose duration depends no longer on the number of tasks involved.

This is an outstanding feature with important implications. On the one hand, low kernel overhead results in faster task response and increased processor schedulability. On the other hand, constant duration of kernel operations makes it possible to precisely estimate task response times taking into account kernel execution effects, which will contribute to higher systems safety and predictability. Ultimately, this technique has made it possible to efficiently implement the timed multitasking paradigm, resulting in jitter-free execution of hard real-time tasks in single-computer and distributed environments.

The presented techniques have been validated in a series of distributed motion control experiments, and jitter-free behaviour has been demonstrated while executing transactions with one-period delay from sampling to actuation, transactions with offsets, as well as distributed transactions with fully decoupled (pipelined) tasks.

References

1. Angelov, C., Sierszecki, K.: Component-Based Design of Software for Distributed Embedded Systems. Proc. The 10th IEEE International Conference on Methods and Models in Automation and Robotics, Medzyzdroje, Poland (2004)
2. Angelov, C.K., Ivanov, I.E., Burns A.: HARTEX - a Safe Real-Time Kernel for Distributed Computer Control Systems. Software: Practice and Experience, vol. 32, No 3, March (2002) 209-232

3. Berthing, J. and Angelov, C.: High-Performance Task Management and Interaction for Safe Real-Time Kernels. CSI Seminar on Safety-Critical Software, Soenderborg (2003)
4. Burns, A., Tindell, K., Wellings, A.: Effective Analysis for Engineering Real-Time Fixed-Priority Schedulers. IEEE Trans. on Soft. Eng., vol. 21 (1995) 475-480
5. Liu, J. and Lee, E.A.: Timed Multitasking for Real-Time Embedded Software: IEEE Control Systems Magazine "Advances in Software Enabled Control", February (2003) 65-75

A New Approach to Deadlock Avoidance in Embedded System

Gang Wu, Zhiqiang Tang, and Shiliang Tu

Computer Science and Engineering Department, Fudan University
220#, Handan Road, Shanghai, PRC 200433
wugamp@yahoo.com.cn
{zqtang, sltu}@fudan.edu.cn

Abstract. Deadlock avoidance algorithms help to improve fault tolerance feature of computer system. Unfortunately, few current algorithms are useful to embedded system either because they only work with single-unit resources, or because the time cost is too high. This paper improves the reduction algorithm with tree structure. According to the new algorithm, only multi-unit resources and the root process in each tree are considered when reducing, while single-unit resources and other processes are ignored because they are combined with tree structure. Thus the new algorithm works with all kinds of resources and its time cost is lower. The new algorithm is useful to embedded system.

1 Introduction

It is almost impossible to build bug-free application software, so fault tolerance feature is important to operating system (OS). Deadlock avoidance algorithms help to improve fault tolerance feature of OS.

As a classical problem, deadlock means that some processes can be blocked forever, waiting for resource impossible to acquire. Those processes will neither respond to anything nor release resources acquired by them. Application software with bug may result in deadlock and the whole system will collapse. If the OS is equipped with some deadlock avoidance algorithm, it will stop improper operation by application software to avoid deadlock, and the whole system will be safe.

Unfortunately, there is no satisfactory deadlock avoidance algorithm for real-time embedded system, yet. The two difficulties to design an ideal deadlock avoidance algorithm are: how to deal with multi-unit resources, and how to decrease the time cost.

Resources can be characterized as containing exactly one unit (single-unit resources) or as containing multiple units (multi-unit resources). Examples of single-unit resources include ports and flag bits. Examples of multi-unit resources include RAM, DMA channels and mailboxes. Some current algorithms [1][2][3] are artful. It's a pity that those algorithms only work with systems without multi-unit resources, while usually an actual system contains multi-unit resources. Those algorithms will make wrong conclusion because of multi-unit resources.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 395-399, 2005.

The only known algorithm that works with multi-unit resources is the reduction algorithm [4] based on wait-for graph (WFG). WFG is defined as a pair <*N*, *E*> where *N* is a set of nodes and *E* is a set of edges. Each node denotes either a process or a resource. Each edge denotes either an assignment relationship or a request relationship between a process and a resource. Each edge $e_i \in E$ is directed between a process node and a resource node. If e_i is directed to a process node then it is an assignment edge, otherwise a request edge. [4]

The reduction algorithm reduces WFG step by step. The unblocked processes are reduced at first, i.e., the process nodes without request edge in WFG are removed. The resources assigned to the processes which have been reduced are released. Since there are more resources available, some originally blocked processes become unblocked and should be reduced consequently. Repeat the routine above until there is no unblocked process left. There is a deadlock if and only if there is one or more than one blocked process left.

The reduction algorithm is not ideal because of its time cost of O(*mn*) complexity. Here *m* is the number of resource nodes, *n* is the number of process nodes, which means that the reduction algorithm is not suitable for real-time system.

The literature [5] also enumerates some other deadlock avoidance strategies, such as heuristics, requiring follow a linear order of resources. These strategies are not ideal because they result in low resource utility-factor and/or unreliability.

In this paper we suggest improving the reduction algorithm with marked tree in order to decrease the time cost and make it more suitable for real-time embedded system.

2 Marked Tree

We use marked tree to combine processes and single-unit resources into a single "virtual process". Each node in a marked tree is a process in fact. We denote two processes with *c* and *p* respectively. *C* is one son of *p* if and only if *c* is waiting for a single-unit resource acquired by *p*. Thus, whether *c* can be unblocked or not is totally decided by *p*. It is also obvious that all processes but the root process are blocked, and whether these processes can be unblocked or not is totally decided by the root process. So a marked tree can be treated as a single process although there may be more than one process contained.

A marked tree is flexible. Each process is a single node marked tree just after it is created. When it requests a single-unit resource acquired by another process, it will become a son of that process.

Each process is marked with a vector. Each scalar in *p*'s vector denotes the amount of a multi-unit resource acquired by all processes in the sub-tree rooted by *p*.

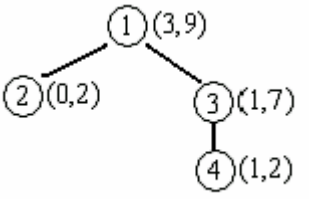


Fig. 1. A marked tree with four processes and two types of multi-unit resources

An example is depicted in figure 1. There are 2 types of multi-unit resources and four processes in this tree. The vector (0,2) appended to process 2 means that:

1) there is no multi-unit resource of the first type acquired by process 2;
2) and there are two units of multi-unit resource of the second type acquired by process 2;

The meaning of the vector (1,2) appended to process 4 is similar.

Process 3 has a son, so its vector shows the total amount of multi-unit resources acquired by both process 3 and 4. For example, process 3 has acquired 5 units of multi-unit resource of the second type, so the second scalar in the vector is 7 (7=5+2).

Similarly, the vector (3,9) appended to the root process indicates the total amount of multi-unit resources acquired by all four processes in the whole tree.

The root process is the only possibly unblocked process in a marked tree. When the root process releases some single-unit resource which is being waited by the sons, one of the sons (this son process is denoted with *s1*) will acquire the resource and become unblocked. The sub-tree rooted by *s1* will break away from the original tree to become a new tree. All other processes waiting for this single-unit resource will become sons of *s1*, thus their sub-tree will shift from the original tree to the new tree.

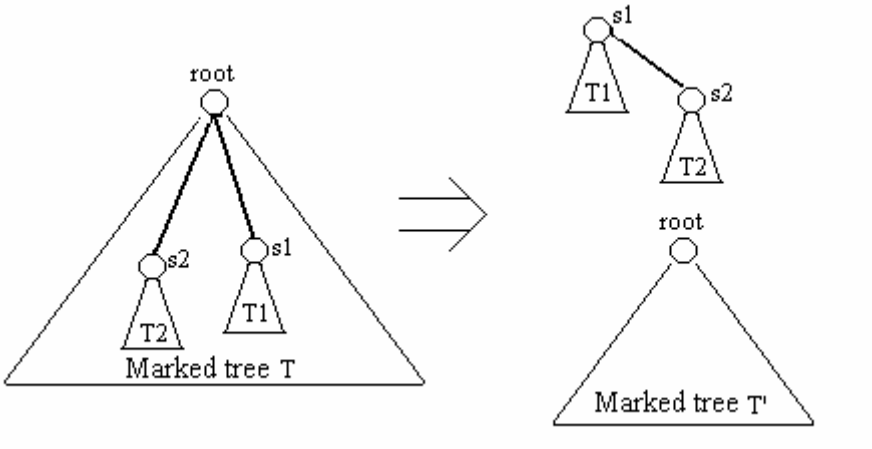


Fig. 2. A marked tree *T* breaks into two

An example is depicted in figure 2. In this figure, the processes *s1* and *s2* are waiting for a same single-unit resource acquired by the *root* process. After the *root* process releases the resource and *s1* acquires it, *T* breaks into two.

Combine marked tree and the traditional reduction algorithm together and we get the new algorithm, which works in this way:

1. When a process *A* requests a single-unit resource which has been acquired by another process *B*, the OS will check if *B* is one descendant of *A* or not. If yes, the request by *A* must be cancelled to avoid deadlock;
2. When process *A* requests some multi-unit resource more than available, the OS will try to reduce as follows (The "*list*" is initialized to contain all root processes including *A*. The "*available vector*" indicates the available amount of each type of multi-unit resource in the system, just like vectors in marked tree):
   (1) Find a reducible process *B* on *list* to reduce. That is, *B* is either unblocked or blocked but its request can be granted by *available vector*;
   (2) Remove *B* from *list*, and add its vector to *available vector*. That is, since all nodes in the marked tree rooted by *B* are reduced, their resources are released and become available to other processes;
   (3) Repeat (1) and (2) until either *list* is empty or there is no reducible process on *list*. If *list* is empty, there won't be deadlock even if *A* is blocked. Otherwise, the request by *A* must be cancelled to avoid deadlock.

The run-time complexity of the first case above is O(*h*) where *h* is the distance from *B* to its root.

The run-time complexity of the second case above is O(*m' n'*). Here *m'* is the number of multi-unit resources and *n'* is the number of marked trees. Since $m' < m$ and $n' <= n$, the new algorithm is faster than the traditional reduction algorithm. Even in the worst case (each process-tree contains only one process, $n' = n$), it is faster because it only takes multi-unit resources into consideration.

3 Implementation

uC/OS is an open source OS mainly for low end real-time embedded system [6]. There is only one type of multi-unit resource in uC/OS: mailboxes. There are also types of single-unit resources in uC/OS, such as ports and flag bits. In order to make uC/OS deadlock-free, the new algorithm based on marked tree is adopted.

A pointer and a vector *v_fgb* are inserted into the process control block (PCB) structure. The pointer points to the father process in marked tree. The vector *v_fgb* contains only one scalar, which indicates the amount of mailboxes acquired by all processes in the sub-tree rooted by the process.

A reduction module is inserted into the OS kernel. The kernel calls this module when a process requests resource but the kernel fails to meet the request. If the reduction module concludes that there will be a deadlock, the kernel will force the process to cancel the request.

Experiments show that no matter how an application programmer tries to make a deadlock in the improved uC/OS, he fails. No obvious influence on CPU time is observed since the system is not too large.

4 Conclusion

The new algorithm is an improvement on the reduction algorithm. It is more efficient and the time cost won't be more than that of the reduction algorithm in worst case. Usually the new algorithm works out sooner. Just like the reduction algorithm, the new algorithm works with multi-unit resources.

The new algorithm helps to improve fault tolerance feature of embedded system, as the system won't collapse just because of bugs in application software.

Acknowledgement

Appreciation to Rod Turner, Victoria University, Australia for his assistance with this paper.

References

1. Ferenc Belik: An Efficient Deadlock Avoidance Technique. IEEE Transaction on Computers, Vol. 39, No. 7, July 1990
2. I. Cahit: Deadlock Detection using (0,1)-labeling of Resource Allocation Graphs. IEE Proc.-Comput. Digit. Tech., Vol. 145, No. 1, January 1998
3. J. Kim and K. Koh: An O(1) time deadlock detection scheme in a single unit and single request multiprocessor system. IEEE TENCON'91, Aug.1991, 219~223
4. Holt R. C.: Some Deadlock Properties of Computer Systems. Computing Surveys, 1972, 4(3), 176~196
5. Gertrude Neuman Levine: Defining Deadlock with Multi-unit resources. Operating Systems Review, ACM Press, 37(3), July 2003, 5~11
6. Jean J. Labrosse : MicroC/OS-II, The Real-Time Kernel. R&D Books, 2002

A Novel Task Scheduling for Heterogeneous Systems

XuePing Ren[1], Jian Wan[1], GuangHuan Hu[2]

[1] Software School, Hangzhou Dianzi University,
Hanghzhou, CHINA, 310018
lilyrxp@hotmail.com
[2] College of Computer Science, Zhejiang University,
Hanghzhou, CHINA
huguanghuan@hotmail.com

Abstract. Heterogeneous computing environments have been widely used in real-time embedded systems. Efficient task scheduling is essential for achieving reliability in embedded systems. In this paper, an algorithm called the Real and Reliable Scheduling (RRTS) has been proposed to improve the system reliability. The RRTS algorithm is achieved by applying a reliability model to a List Scheduling algorithm. The algorithm firstly calculates Earliest-Finish-Times (EFTs) and reliability costs of a task in every processor. Then the minimum EFT is selected. Finally, the task is scheduled to a processor according to the EFT and reliability costs. The experimental results and examples show that the algorithm can improve system reliability significantly while maintaining high performance.

1 Introduction

Heterogeneous distributed systems have been increasingly used for scientific, commercial and military applications. To develop these heterogeneous embedded systems in critical or military applications, an efficient scheduling approach is needed to satisfy the requirements of both performance and reliability.

Topcuoglu *et al*. propose two algorithms to schedule tasks in a heterogeneous environment in [1] and [2]. Both of them use the critical path to prioritize tasks first, and then schedule them. In [3], a scheduling algorithm based on dynamic critical paths is discussed. In general problems of task scheduling, the scheduling of a DAG determines that the start time of each node must satisfy the dependency relations in the DAG [4]. In [5], a task-scheduling algorithm (DBLF) for real-time heterogeneous embedded systems is introduced. The DBLF algorithm selects the ready task with maximum blength (n_i) at each step and assigns the selected task to a suitable processor. The processor satisfies the precedence sequence and has minimum earliest-finish-time (EFT) of the task. This algorithm focuses on scheduling length. However, in the above algorithms, little attention has been paid to the reliability issue.

Recently, some research work has begun to address the scheduling problems with reliability optimization. In [6], a reliability model for general heterogeneous computer systems is established and several algorithms to solve the scheduling problem using this model are investigated. In [7], two algorithms focus on task scheduling and

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 400-405, 2005.

resource allocation with reliability maximization in heterogeneous systems. But both of them are based on ALAP, and an important parameter, which stands for the criticality of node in reliability model, is almost ignored. Those have a great influence on the availability of algorithm. In this paper, a reliability model is applied to a List Scheduling algorithm, and a new algorithm is proposed. The system reliability can be maximized and high performance can be achieved in our algorithm.

2 System Characteristics

A Directed Acycline Graph (DAG) is used to model an application. A DAG G={U, E, C, sourcenode, sinknode} is a node-weighted and edge-weighted directed graph; where U={u_1, u_2 …u_n} is the set of tasks, $E \subseteq U * U$ is the weighted edge set that defines the precedence relations among tasks in U; the weight on each edge, e_{ij} represents the time of inter_processor communication between two tasks; C={c_1, c_2 …c_n} is the set of criticality of the corresponding task nodes. Criticality is the parameter to represent how critical a node is. The higher its criticality is, the more critical a task is. Sourcenode is the only entry task and sinknode is the only exit task for one graph.

A heterogeneous system consists of a certain number of heterogeneous processor elements (PEs). Assume we use a set P={P_1, P_2 … P_m} to denote all its PEs. A matrix D is used to represent the communication delay among all processors, where d_{kb} represents the delay involved in sending a message of unit length from P_k to P_b. And $t_j(i)$ denotes the computation time of u_i on P_j

Definition 1. Schedule length is finish time of sinknode, after all tasks are scheduled in the graph.

Definition 2. The initial execution time of task is the average execution time in all processors. If u_i is assigned to P_j, $R_i = t_j(i)$

$$Ini\ (R_i) = \underset{q \in P}{average}\ \{t_q(i)\} \tag{1}$$

Definition 3. B*length* (u_i) is the longest path length from u_i to sinknode (include u_i execution time). $succ\ (u_i)$ is the direct subsequence set.

$$\left.\begin{array}{l} blength\ (\sin knode) = R_{\sin knode} \\ blength\ (u_i) = \max\limits_{u_j \in succ\ (u_i)} \{blength\ (u_j) + R_i + w_{ij}\} \end{array}\right\} \tag{2}$$

Definition 4. Criticality: $c_i = \text{blength}(u_i) / \max\limits_{n_j \in U} (\text{blength}(u_j))$ (3)

In this paper, reliability is defined as the probability that the system will not fail during the time that it is executing tasks. The probability of the system not to fail is:

$$pr = \prod_{j=1}^{M} \prod_{i=1}^{N} (1 - f_j)^{X_{ij} t_j(i)} \times \prod_{k=1}^{M} \prod_{b=1}^{M} \coprod_{i=1}^{N} \coprod_{j=1}^{N} (1 - g_{kb})^{X_{ik} X_{jb} W_{ij} d_{kb}} \tag{4}$$

We suppose that f_j is the failure rate of PE P_j, g_{kb} is the failure rate of the communication link from P_k to P_b, W_{ij} is the volume of data that task u_i needs to send to task u_j and s be communication speed, so $W_{ij} = e_{ij} * s$.

To calculate the impact of each task to the system reliability cost when it is scheduled on a PE, Rc_{ij} is used to denote the reliability cost for u_i to be scheduled on P_j. Rc_{ij} can be expressed as follows (more detail can be found in [7]):

$$Rc_{ij} = f_j c_i t_j(i) + \sum_{p \in Pred(i)} \sum_{k=1}^{M} g_{kj} X_{pk} W_{pi} d_{kj} \quad (5)$$

where pred (i) is the set, which includes all u_i's predecessors. Therefore:

$$RC = \sum_{i=1}^{N} \sum_{j=1}^{M} X_{ij} Rc_{ij} \quad (6)$$

In order to maximize system reliability, we need to minimize the Reliability Cost. We will use the "Reliability Cost" as the indicator of how reliable a given system is when a group of tasks are assigned to it. The lower the reliability cost is, the higher the reliability is.

3 Scheduling Algorithm

In this section, we apply the reliability model to a List Scheduling algorithm, an algorithm: the Real and Reliable Task Scheduling (RRTS) is developed. We define the problem of heterogeneous reliability scheduling as: given a heterogeneous system with m PEs, a DAG G and a time constraint L, finds a task schedule for G such that the Reliability Cost is minimized and high performance is achieved within L. The RRTS algorithm is proposed to solve the problem. We formalize following rules for task scheduling.

Rule 1. A node u_i can be inserted into a processor P_j.

$$\text{Avail}(u_i, p_j) = \max\{schedule(u_k, p_j) + R_k\} \quad (7)$$

where schedule $(u_0, p_j) = 0$, and schedule $(u_{m+1}, p_j) = \infty$, if there exists some k such that $k = \min_{0 \le l \le m}\{schedule(u_{l+1}, p_j) - schedule(u_l, p_j) - R_l \ge w(u_l, p_j)\}$

Rule 2. $CFT(c_{pi})$ represents communication-finish-time between n_i and its precedence u_p. Insert (u_i, p_j) represents the start time of u_i inserted to P_j. EFT (u_p) represents the earliest finish time of u_p. If $P_{u_p} = P_{u_i}$, $r_1 = 0$, else $r_1 = 1$.

$$\text{Insert}(u_i, p_j) = \max\{\max_{u_p \in pred(u_i)}\{\text{EFT}(u_p), r_1 * CFT(c_{pi}) + c_{pi}\}, \text{Avail}(u_i, p_j)\} \quad (8)$$

$$CFT(c_{pi}) = \max\{\text{EFT}(u_p), \text{avail}(c_{pi}, F(P_{u_p}, P_{u_i}))\} \quad (9)$$

Rule 3. Look-ahead strategy. Let u_c be the direct subsequence of u_i in the critical path.

$$\text{EFT}(u_i, p_j) = \text{Insert}(u_i, p_j) + w(u_i, p_j) + \text{insert}(u_c, p_j) + w(u_c, p_j) \quad (10)$$

By using this looking-ahead strategy to examine the start time of critical children, the proposed algorithm can avoid scheduling a node to an inappropriate processor. As a result, it avoids the danger of increasing the schedule length in subsequent steps.

Rule 4. $\mathrm{EFT}(u_i, p_x) = \min_{p_J \in P} \{\mathrm{EFT}(u_i, p_j)\}$ (11)

If difference **between** EFT (u_i, p_x) and one EFT (u_i, p_j) is less than minf
Then task u_i is scheduled to a processor according to corresponding reliability costs and minR.
Else task u_i is scheduled to processor p_x.

Where j<>i, minf and minR are determined beforehand according to actual requirement of system. Using Rule 4, we ensure that system can effectively reduce reliability cost. As a result, system can achieve the higher performance and the less reliability cost. The asymptotic complexity of our algorithm is O (n^3), which is equal to that of DBLF. DBLF has good performance without discussing the reliability issue.

Algorithm: RRTS
Compute the blength of all tasks;
While not all tasks have been scheduled **Do**
Select the ready nodes u_l with maximum *blength* (u_l);
For each processor p_k in the processor-set P **Do**
Insert the task u_l into processor p_k
Compute corresponding E*FT* and *R*
End-For
$\mathrm{EFT}(u_i, p_x) = \min_{p_J \in P} \{\mathrm{EFT}(u_i, p_j)\}$
For each processor p_j in the processor-set P **Do**
eft=|EFT(u_l, p_x) -EFT(u_l, p_j)|
If ((eft = =0) or (**eft**<minf and $|Rc_{lx} - Rc_{lj}| \geq$ minR))
Then u_l is scheduled to p_y, which has less Rc. mark u_l scheduled and exit.
End-For
If (u_l is not marked**)**
Then u_l is marked and scheduled to p_x.
Adjust the ready tasks set.
Compute the blength of all tasks according to the *schedul*e (u_l, p_j);
End-while with not all tasks scheduled

---

4 Example

To illustrate the effectiveness of the proposed algorithm, we use an example task graph. Among all real-time heterogeneous algorithms, DBLF has best performance (more detail can be found in [5]). For comparison, the schedules generated by DBLF are also presented.

We describe the example in Figure 1. Assume there is a heterogeneous system that consists of 3 heterogeneous PEs. An exemplary DAG is shown in figure 1(a). The failure rate of the communication links between the PEs is 0.001 for both directions. The communication delay between the PEs is shown in figure 1(b). The failure rates of PEs are shown in figure 1(c). Interconnect architectures among 3 PEs are shown in figure 1(d). The execution times of each node for different PEs are shown in figure 1(e). The criticality of all nodes is computed according to definition 4.

In the example, schedule length generated by DBLF and RRTS is completely the same, as two schedules for the DAG are shown in Fig. 2 and Fig.3. In Fig. 2, its system reliability cost is 8 by DBLF. In Fig. 3, task u5 is scheduled to processor p2, so its system reliability cost is reduced to 3.4 by RRTS. Both schedules satisfy the time constraint while the latter has a lower reliability cost. We reduce 57.5% reliability cost without lost performances. This example shows that RRTS produces less reliability cost and achieves high performance at the same time.

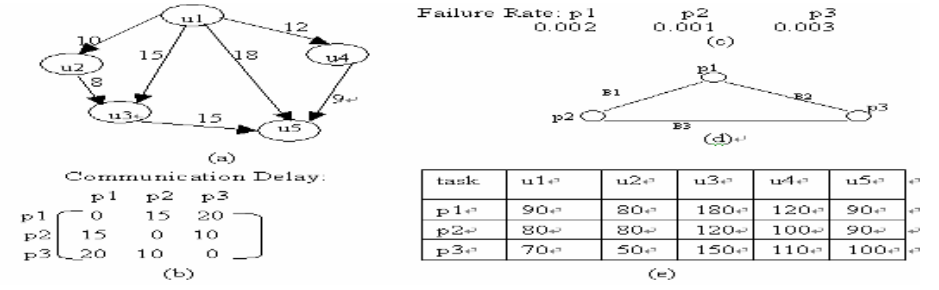


Fig. 1. (a) A DAG. (b) Communication delay between the PEs. (c) Failure Rate of PE. (d) Interconnect architectures Among 3 PEs. (e) Computation times of tasks

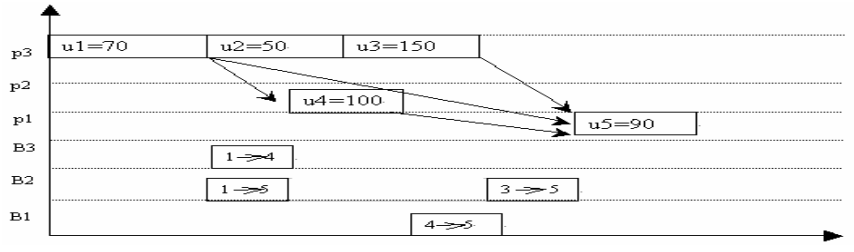


Fig. 2. The schedule of the fig.1 task graph generated by DBLF

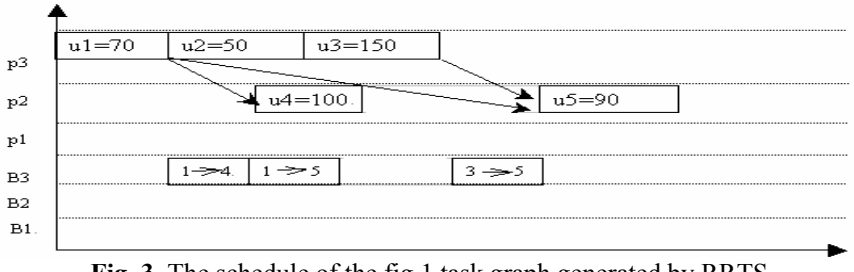


Fig. 3. The schedule of the fig.1 task graph generated by RRTS

5 Experiments and Conclusion

To evaluate the effectiveness of our algorithm, we simulate the heterogeneous environment and generate several random DAGs in experiments. Two main measurements to evaluate the scheduling algorithms are the schedule length and the

reliability cost. The comparison of the results is shown in Table1. As a result, the average fall of Reliability cost is 18.75% while the average increase of schedule length is 3 % in experiments.

Table 1. Comparison of schedule result between DBLF and RRTS algorithm

Tasks	20				30			
PEs	4	8	12	16	4	8	12	16
Schedule length add	3%	0	4%	6%	1%	3%	0%	6%
Reliability cost less	26%	18%	19%	22%	17%	15%	21%	12%

We propose a scheduling algorithm, the Real and Reliable Scheduling algorithm (RRTS). The example and experiments all prove our approach is effective. Simulation results show that the algorithm improves the system reliability significantly over the DBLF while maintaining high performance. RRTS gives best reliability and high performance in any situation.

References

1. Haluk Topcuoglu, Salim Hariri, and Min-You Wu, "Task scheduling algorithms for heterogeneous processors," in *Proceedings of the 8th Heterogeneous Computing Worksho*p, 1999, pp. 3–14.
2. Haluk Topcuoglu, Salim Hariri, and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed System*s, vol.13, no. 3, pp. 260–274, March 2002.
3. Yu-Kwong Kwok and Ishfaq Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed System*s, vol. 7, no. 5, pp.506–521, May 1996.
4. Giovanni De Micheli, *Synthesis and Optimization of Digital Circuit*s, McGraw-Hill, New York, 1994.
5. QIU Wei-Dong, CHEN Yan, LI Jie-Ping, PENG Cheng-Lian, "A Task Scheduling Algorithm for Real-Time Heterogeneous Embedded Systems", *Journal of Softwar*e, vol.15, No. 4, pp.504~511,2004.
6. Sol M. Shatz, Jia-Ping Wang, and Masanori Goto, "Task allocation for maximizing reliability of distributed computer systems," *IEEE Transactions on Computer*s, vol. 41, no. 9, pp. 1156–1168, September 1992.
7. Yi He, Zi Li Shao, Bin Xiao, Qingfeng Zhuge, Edwin Sha," Reliability Driven Task Scheduling for Heterogeneous Systems," in www.utd.edu /~edsha/papers/ YiHe/.

Applying Component-Based Meta-service in Liquid Operating System for Pervasive Computing[1]

Bo Ma, Yi Zhang, and Xingguo Shi

Laboratory for Internet Software Technology
Institute of Software, Chinese Academy of Sciences
No.4 South Fourth Street, Zhong Guan Cun, Beijing, China
{mabo, zhangyi, shi}@itechs.iscas.ac.cn

Abstract. Liquid meta-service, a component-based operating system layer, is intended to enable embedded operating system to support pervasive computing by meeting the requirements of being spontaneous and adaptive. This is accomplished through two key features: (1) configurable, component-based infrastructure services called meta-service, and (2) a meta-service components trading service, which provides selecting and configuring mechanism for meta-service components. In this paper, we discuss the design principle, component model, implementations of some meta-service, and analyze the performance impact of using meta-service in embedded operating system.

1 Introduction

Pervasive computing raises a number of new challenges for application and the underlying operating system. Effective use of smart spaces, minimal user distraction, localized scalability and uneven conditions masking are recognized as the characteristics of pervasive computing [5]. They require pervasive application to be a) adaptive. Because the availability of underlying services keeps varying and pervasive application keeps migrating from one computing context to another, and b) spontaneous, that is to say, being able to dynamic discovering, adapting, connecting and coordinating disparate software units to form working and reliable pervasive applications.

There are two general approaches to implementing software adaptation [10]. *Parameter adaptation* modifies program variables that determine behavior, such as the Internet's Transmission Control Protocol (TCP). But parameter adaptation does not allow new algorithms and components to be added to an application after the original design and construction. This is unacceptable for pervasive applications for its mobile and spontaneous nature. By contrast, *compositional adaptation*, which exchanges algorithmic or structural system entities with others that improve a program's fit to its current environment. The composability, reusability and

[1] Supported by the National High-Tech Research and Development Plan of China under Grant No.2002AA1Z2302

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 406-411, 2005.

configurability of software components make it a good candidate technology to implement compositional adaptive for pervasive application.

To tackle the problem of spontaneous in operating system level, we propose an approach of using trading service in the kernel. Trading is the natural mechanism defined in object- and component-based systems for searching and locating services. Context-aware software, such as pervasive computing, can greatly benefit from trading since it provides service discovery in local environments and enables automatic application re-contexting [12].

Liquid operating system is designed as a platform for embedded networked devices. Liquid meta-service is a component-based layer in the Liquid kernel. Liquid meta-service enables Liquid to support pervasive computing by providing necessary infrastructure components and a meta-service components trading service.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the design of Liquid meta-service. Section 4 describes some meta-service in detail, and presents the performance analysis of using meta-service. Section 5 draws some conclusions and presents future work.

2 Related Work

In pursuing the solution with component technology in the operating system level, two critical choices have to be made. First, to what degree the operating system is made into components, and second, what granularity these components are. They are two sides of one coin.

Currently, most component-based operating systems take the approach of making the entire system into components and keep the granularity of components relatively fine-grained, as exemplified by PURE [2], Pebble [3]. Modularity achieved by isolation comes at a price, partition of system into smallest components to maximize reuse conflicts with efficiency. The granularity of Liquid meta-service keeps balance between modularity and performance.

Task-oriented computing [11] defines a skeleton of infrastructure for pervasive computing. Liquid meta-service components trading service can be employed as the task assembly engine, thus the trading service presented in this paper can be used as a complementation of task-oriented computing. General introduction of using trading service in COTS development can be found in [12].

3 Design of Liquid Meta-service

3.1 Liquid Meta-service Principle

The principle of Liquid meta-service consists of the following four key ideas. When combined, they enable our goals.

- *The operating system services are divided into two categories, a) meta-service, which consist of a group of meta-service components, and b) ordinary service.*

Among operating system services, some have more effect on the support of pervasive computing than others on the aspect of meeting the levity requirements of infrastructure, such as scheduling (may switch from time-sharing to real-time), network communication (may switch from synchronous to asynchronous) and quality of service (QoS) ensuring.

- *Meta-service is implemented as a separate portable layer, which can run either as an integrated part of the Liquid kernel, or on conventional operating systems.*

Making meta-service a portable layer, we broaden its applicability so that it can be adopted by conventional operating systems, which is meant to support pervasive computing in system level.

- *Each meta-service is designed as an extensible component framework [8] by using a role-based paradigm [4]. Each role is specified by contract [9], and is played by a specific kind of meta-service component.*

Each meta-service can accept dynamic insertion of component instances, and replace them at runtime. By forcing meta-service component instances to perform certain tasks via mechanisms under the control of each meta-service, meta-service can enforce specific polices and architectural principles.

- *A trading service is responsible for configuring each meta-service with suitable meta-service components when it accepts the infrastructure requirements raised by pervasive application.*

3.2 Liquid Meta-service Component Model

According to the principle described above, a component model for Liquid meta-service component is conceived. The component model specifies how meta-service components are structured and composed. Figure 1 shows an overview of the component model.

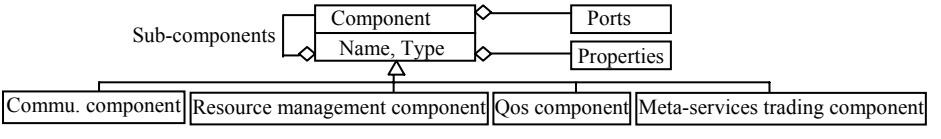


Fig. 1. Meta-services component model

A meta-service component is a computational element with a name, a bundle of properties and ports, and a behavior. Ports are used to represent components' data to others. Each component may have multi provides-ports and requires-ports. Components with compatible conjugated ports can be connected [1]. Behavior of a component consists of a procedure that reads and writes data available at its ports, and may produce effects in the physical world; the actual behavior is hidden in the implementation of the component, and not shown in the figure. Components may contain one or more sub-components.

4 Meta-service Example and Performance Analysis

In this section, we present two meta-services in detail to illustrate how we design and implement the features described above. Then the performance analysis is presented.

4.1 Scheduler Meta-service

Traditional scheduling mechanisms are limited when dealing with pervasive computing: (1) the schedule strategy is fixed, that is, either schedule algorithms or performance parameters are fixed, and (2) the schedule strategy is configured statically before the system boots up, and makes no response to keep changing scheduling requirements raised by pervasive application, such as switch from time-sharing to real-time scheduling.

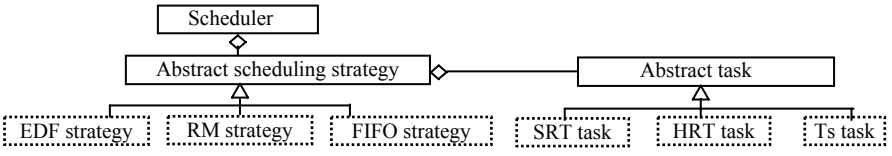


Fig. 2. Structure of scheduler meta-service

We tackle the problem with the scheduler meta-service. Its structure is showed in figure 2. The schedule strategies are represented as roles (dash line rectangle) to be played by scheduling meta-service component instances selected by trading service. As the component framework for this meta-service, the scheduler supervises all schedule-strategies. Each strategy component contains a task group (a queue, a stack or a pool) of its own. As soon as a new computing task is submitted by pervasive application, the scheduler asks the trading service to select a suitable schedule strategy component from the repository according to the characteristics of the task, and puts it into the task group.

4.2 Meta-service Component Trading Service

In Liquid, we apply trading in the support of pervasive computing. There is a trading meta-service on each Liquid node. It is the portal for pervasive application to interact with Liquid. To meet the infrastructure requirements dynamically, a pervasive application gathers the infrastructure requirements in terms of meta-service as soon as it comes into being in a given context through ad hoc networking [6], and feeds the requirements to the trading meta-service. According to the requirements, the trading service selects candidate meta-service components from the repository, configures them, and submits the components to various meta-service. Then, each meta-service configures its framework by placing the components at the right roles. This trading-centric architecture of meta-service layer is inspired by *Façade* design pattern [7].

```
<?xml version="1.0"?>
<LiquidMetaServicesComponent
      name="EDFScheduler"
   type="resource management">
          <ports>
       <provides-ports>
<port name="addTask">…</port>
       </provides-ports>
       <requires-ports>…
       </requires-ports>
          </ports>
      <property-bundle>
    <property>…</property>
      </property-bundle>
<sub-components>…</sub-components>
 <LiquidMetaServicesComponent>
```

Fig. 3. A simple meta-services component descriptor

```
<?xml version="1.0"?>
<LiquidMetaServicesRequirements
 name="LiquidWeatherStationReq">
 <ResourceManagement>
  <Scheduling type="softRT"
     threshold="2sec">…
     <property-bundle>
      <property>…</property>
     </property-bundle>
  </Scheduling>
 </ResourceManagement>
 <Communicatoin>…</Communicatoin>
 <QoSReq>…</QoSReq>
</LiquidMetaServicesRequirements>
```

Fig. 4. A simple meta-services requirement

The trading meta-service consists of a group of trading meta-service components; each component is responsible for trading one category meta-service. On booting up, all meta-service components register with the corresponding trading meta-service component by passing a component descriptor in XML format. Figure 3 shows a simple skeletal instance. The infrastructure requirement is also documented as XML file as shown in Figure 4. The trading meta-service uses a backtracking algorithm to select candidate components whose properties are compatible with the corresponding requested properties, and then match these components against roles in each kind of meta-service to form a valid configuration.

4.3 Performance Analysis

We conduct some experiments to evaluate the performance impact caused by meta-service on ordinary computing tasks. A group of compute-intensive tasks (matrix multiplication) are executed concurrently under the scheduling of Linux first, and then run as tasks for scheduler meta-service. Table 1 shows some results we have gained on Linux running on Pentium III 1.2G CPU with 256M memory. Numbers in the first row indicates the task number; data in the table are the average execution time (in seconds) under different running conditions. The average performance of scheduler meta-service reaches 92.2% of Linux, results of other meta-service range from 80% to 93% of Linux. The system can be used in a full version of meta-service with slight performance degradation.

By applying the principle we identify in section 2, we achieve a fair balance between modularity and efficiency as shown by the performance analysis results.

Table 1. Some results of performance tests of scheduler meta-service

Time(sec)	1	2	3	4	5	6	7	8	9	10
Linux	1.54	2.83	4.22	5.57	6.94	8.31	9.50	10.79	12.23	13.43
Scheduler	1.70	2.93	4.30	6.00	7.06	9.39	11.32	11.69	13.63	14.73
L / S	0.90	0.96	0.98	0.92	0.98	0.88	0.84	0.92	0.89	0.91

5 Conclusion and Future Work

Pervasive computing is an important and difficult problem for the embedded system design. In order to tackle the problem in system level, we propose an approach to build a component-based meta-service layer in the embedded operating system. It comprises (1) a group of configurable, component-based infrastructure services, and (2) a trading service for meta-service components. With meta-service, the Liquid operating system is capable of supporting pervasive computing by meeting the levity infrastructure requirements raised by pervasive application.

The ongoing research includes extending the required meta-service specification with priority, which enables the users to specify the importance of each sub-requirement more precisely. Extra-functional properties and user preference are not covered by our trading service. These can be used to form heuristic functions for selecting, matching, and closing processes.

References

1. Ommering, R., Magee, J. J.: The Koala Component Model for Consumer Electronics Software. IEEE Computer, Vol.33, No3 (2000) 78-85
2. Beuche, D., Guerrouat, A., Papajewski, H., Schroder-Preikschat, W., Spinczyk, O., Spinczyk, U.: The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems. In: Proc. of ISORC'99, St Malo, France (1999) 45-53
3. Gabber, E., Bruno, J., Brustoloni, J., Silberschatz, A., Small, C.: The Pebble Component-Based Operating System. In: Proc. of the USENIX Annual Technical Conference, Monterey, CA, USA, June 6-11 (1999) 267-282
4. Riehle, D.: Framework Design: A Role Modeling Approach. Ph.D. Thesis, No.13509. Zrich, Switzerland (2000)
5. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications, August (2001) 10-17
6. Feeney, L. M., Ahlgren, B., Westerlund, A.: Spontaneous Networking: An Application-oriented Approach to Ad Hoc Networking. IEEE Communications Magazine, Vol.39, No.6, June (2001) 176-181
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Massachusetts (1995)
8. Szyperski, C., Gruntz, D., Murer, S.: Component Software. Addison Wesley (2002)
9. Helm, R., Holland, I. M., et al.: Contracts: Specifying Compositions in Object Oriented Systems. In: Proc. of OOPSLA/ECOOP'90, ACM SIGPLAN Notices (1990) 169-180
10. McKinley, P. K., Sadjadi, S. M., Kasten, E. P., Cheng, B. H. C.: Composing Adaptive Software. IEEE Computer, July (2004) 56-64
11. Want, Z., Garlan, D.: Task-Driven Computing. Technical Report, CMU-CS-00-154, School of Computer Science, Carnegie Mellon University (2000)
12. Iribarne, L., Troya, J. M., Vallecillo, A.: A Trading Service for COTS Components. The Computer Journal, 47(3) (2004) 342-357

Embedded Operating System Design: The Resolved and Intelligent Daemon Approach[1]

Hai-yan Li, Xin-ming Li

ZhuangBei Institution of Technology, Beijing, China
lhycya@21cn.com

Abstract. Embedded system with the kernel of embedded operating system becomes the ubiquitous computing unit now. Embedded operating system should be of hard real-time, high reliability and high availability. Aiming at satisfying the demands of applications, based on analyzing the existing three kinds of operating system architectures and some new design technologies, this paper presents a novel design of operating system which is based on resolved and intelligent daemon and agent-based coordination managing components.

1 Backgrounds

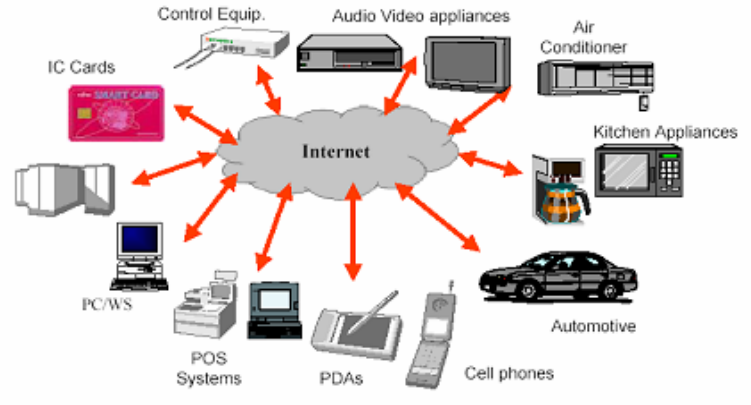


Fig. 1. Computing Everywhere

Demands of application and progress of technology drive the development of embedded operating system. Especially in recent ten years, with the development of network technology, chip technology, human-computer interaction technology and software designing technology, the development speed of embedded operating system has been accelerated greatly, which enables embedded operating system to meet the

[1] This work has been supported by 863 program of China, grant no. 2003AA1Z2050

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 412-417, 2005.

abundant application demands day by day, and has promoted deepening and popularization of embedded system. As Fig.1 shows, the embedded system with the kernel of embedded operating system becomes the ubiquitous computing unit.

Our research will be used in co-ordination control information processing subsystem in distributed spacecraft system. Distributed spacecraft system is made up of several micro spacecrafts which cooperate to accomplish some tasks.

2 Analysis of Operating System Architecture and Design Technology

Because more and more functions and characteristics are added to operating system, at the same time the hardware platform becomes more and more complicated and diversified, the scale and complexity of operating system increase notably. However, the expanding of operating system scale causes three general problems: (1) function of operating system becomes more difficult to be expanded. It needs more time to develop a new operating system or upgrade and reconstruct an old operating system; (2) relationship among numerous modules is difficult to identify. Operating system has potential errors, and it's very difficult to find out and eliminate these errors; (3) performance can't always satisfy the demand of application.

In order to solve the problems mentioned above, researchers have carried on a large amount of research on architecture and designing technology of operating system for many years, and many research fruits have promoted development of operating system technology and further popularization of computer.

Operating systems are broadly classified as three categories.

(1) Monolithic Operating System. This approach is well known as "The Big Mess". The operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs. DOS is a typical representative of this kind of operating system.

(2) Layered Operating System. Operating system is divided into a number of layers, each built on top of lower layers. The bottom layer is the hardware; the highest is the user interface. With modularity, layers are selected such that each uses functions and services of only lower-level layers. In this design, the abstraction and resource management code of the operating system are all placed into the operating system kernel. UNIX is the famous example of this kind of operating system.

(3) Client/Server or Microkernel Operating System. In this design, abstraction and resource management code is moved as much as possible outside of the kernel in the interest of modularity. The interface among components of the system then becomes message passing from the application to whatever component providing the desired service. These components run as separate programs on the same operating system. These system programs are called servers, and applications are their clients. Some examples of microkernel architecture include Mach and QNX.

Generally speaking, operating system kernel of traditional architecture is becoming larger and larger. The architecture has already been very complicated. Function is difficult to be expanded and customized. System is difficult to be upgraded. The

essential reason is that designers often sacrifice the performance of some frequently used systematic functions to satisfy the demands of a small amount of less used functions. Conversely, enhancement of system complexity has increased the risks of system unreliability and unsafety. A lot of researches, such as distributed operating system, security operating system, real-time operating system, object oriented operating system, have made a lot of achievements, but such kinds of problems exist in practical applications. We believe that function and structure of the modern operating system restrict development of operating system.

With the development of operating system technology, various kinds of new technologies begin to be applied to the design of operating system too, for instance object oriented technology, component-based technology and agent-based technology and so on.

(1) Object oriented technology. An object is an explicit software unit, which includes a set of relevant data and processes. Usually, these data and processes are not directly visible outside the object, and there is a set of interface for other software to access the data and processes.

(2) Component-based technology. Component is the reusable software component which can be used to construct other software. It can be the form of encapsulated object class, class tree, functional module, software framework, software architecture, document, analysis component, design pattern and so on. Component can be classified as component class and component instance. Application software can be built through assembling and controlling component instance.

(3) Agent-based technology. Agent is an entity which resides in the environment. Agent-based software development method is a kind of abstract description form which is used to describe complex parallel system. Its main idea is to regard computer system as a society composed of agents, and introduce the concepts of psychology and anthropology to understand and implement computer system.

3 Design of Operating System Based on RID

Embedded operating system is developing towards the direction of networked, intelligent, low consumption, hard real-time, high trusted and high availability. Aiming at new computation modes such as autonomous computation and pervasive computation, after deep analysis and research on architecture and design technology of many kinds of operating system, we put forward a new operating system design based on RID (Resolved and Intelligent Daemon).

The key to RID has two points. One is "resolved" principle. Similar to the hardware "RISC" technology, operating system kernel only supports the simple and necessary mechanism. Complicated functions or tactics are offered through "making up" on the basis of the mechanism that RID offers. The other is "intelligent" principle. Some less used functions are only loaded when applications need. At the same time, operating system based on RID can optimize itself and auto-configure according to the application and environmental characteristic.

Logically, an information process system is made up of three cells: computation cell, memory cell and human-computer interaction cell. As to computer hardware system, the basic components are in mainly three parts: CPU, memory and I/O

system. On the one hand, operating system separates hardware from application, which makes the renewal of hardware and change of application will not influence each other. On the other hand, operating system is a bridge between hardware and application, through which application can access hardware resources. Operating system should offer suitable support according to special demands and performance for every application. However, there exists contradiction: if we develop an operating system only aiming at one special application and hardware environment, application effect should be good theoretically, but the development cost is very high, and the system dependability may not been improved either. If we make use of a riper operating system to support new applications, the cost is much lower, and the dependability of operating system is much higher, but the effects of some applications are relatively bad. Therefore, an ideal embedded operating system should hide the impact of hardware change on applications, coordinate multi-application access to hardware resources. A very important aspect is that operating system should offer suitable support according to application background and hardware environment, in order to reach the best ratio of cost performance.

As Fig.2 shows, logically, embedded operating system can be divided into three layers: the upper layer is AFL (Application-specified Function Layer), the middle layer is CFL (Common Function Layer), and the bottom layer is HSL (Hardware Shield Layer).

AFL is a set of functions correlated with application to support all kinds of applications. On the one hand, CFL utilizes HSL to manage hardware. On the other hand, CFL offers support for AFL. HSL offers support for CFL and AFL according to hardware of all kinds.

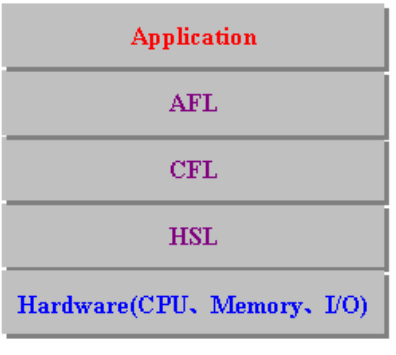


Fig. 2. Logic Architecture of Embedded Operating System

According to the design of RID, operating system is divided into two major parts: RID and CMC (Coordination Managing Component). Fig.3 shows software architecture based on RID.

The separation of policy from mechanism is a very important principle, which allows maximum flexibility if policy decisions are to be changed later. In traditional operating system, mechanisms and policies are usually all integrated in kernel, but in our design only mechanisms reside in kernel. Policies, such as real-time scheduling policies, memory management policies, are offered by CMC. It is favorable to system flexibility and expansibility, convenient for customization according to application.

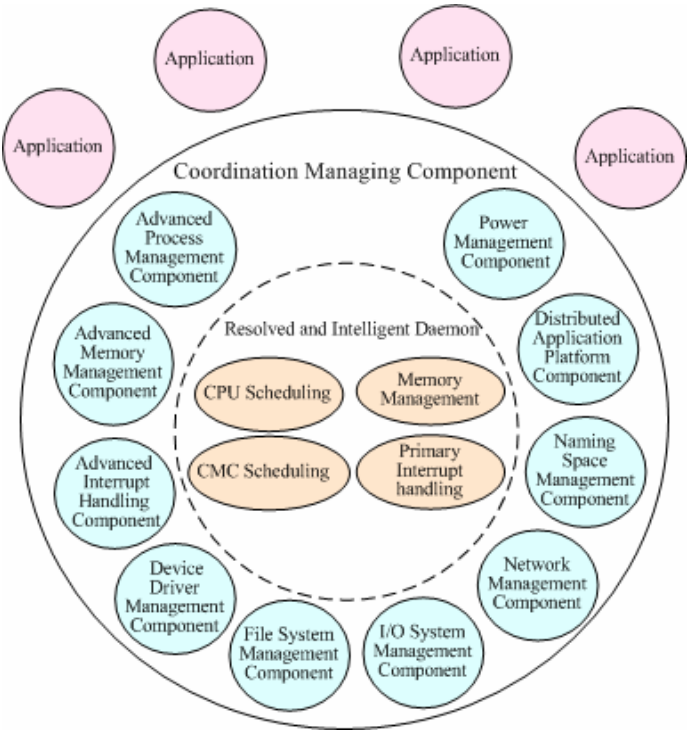


Fig. 3. Software Architecture Based on RID

RID is only responsible for CPU scheduling, memory management, primary interrupt handling and CMC scheduling. In CPU scheduling, there is only schedule mechanism without schedule policies. In memory management, it only maps logical address to physical address. In primary interrupt handling, it only accepts and identifies hardware interrupt. In CMC scheduling, it only loads and unloads CMC.

As to the address space management, we adopt sliding address window technology. Sliding address space is that, working space of an execution entity is a certain area of a virtual address, and its size is confirmed according to its demand. By adopting this kind of technology, cost of process switch can be reduced, and utilization ratio of memory resources can be raised. Foundation of sliding address window technology is single address space technology. All processes operate in a same address space and different process is in different area in the virtual space, this is the thought of SASOS (Single Address Space Operating System).

4 Agent-Based Design of CMC

Each CMC is an agent. It's convenient for modification and expansion of CMC functions, without recompilating kernel and even restarting the system. CMC should be designed of single function and modularization. Interaction among CMCs should be little. All CMCs are independent in binary level. RID and CMCs communicate

through well defined interfaces, so do one CMC and other CMCs. They do not need to find out the realization details of the others.

CMCs are made up of ten parts.(1) Advanced process management component. It provides policies of when to schedule and what to schedule. (2) Advanced memory management component. It provides algorithm of memory allocation and release, and protects adress space. (3) Advanced interrupt handling component. It provides interrupt service routines to handle interrupt which is accepted and identified by RID. (4) Device driver management component. It manages device drivers and provides device driver communication mechanism of I/O request and IOCTL. (5) File system management component. It handles requests of file open, close, read and write. (6) I/O system management component. It manages all I/O devices of computer, handles device errors and provides convenient interfaces for other parts of system to use devices. (7) Network management component. It manages all network devices and provides a set of uniform interfaces for other processes to establish network connect. (8) Naming space management component. It manages name space for operating system to implement accessing transparency, position transparency and transfer transparency etc. (9) Distributed application platform component. It provides distributed application mode of application task or service level. (10) Power management component. It manages power supply to reduce energy consume and prolong use period of battery.

5 Conclusions

Embedded systems, such as distributed spacecraft system, wireless micro-intelligent sensor networks system and system in vehicles, require high dependability and real-time performance. We have analyzed and researched the demands of application backgrounds and characteristics of embedded operating system, and then put forward a new design of operating system based on RID.

References

1. Andrew.S.Tanenbaum: Modern Operating Systems. China Machine Press, Beijing (2001) 18–19
2. Michael Wooldridge: An Introduction to MultiAgent System. Publishing House of Electronics Industry, Beijing (2003) 15–18
3. Wilkinson.T, Murray.K, Russell.S: Single address space operating systems. Technical Report, UNSW-CSE-TR-9504, Sydney: University of NewSourthWales (1995)
4. Chase.J, Feeley, M,Levy.H: Some Issues for Single Address Space Systems. In: Proceedings of the 4th IEEE Workshop on Workstation Operating Systems. LosAlamitos, CA: IEEE Computer Society Press (1993) 150–154

New Approach for Device Driver Development – Devil+ Language

Yingxi Yu[1], Mingyuan Zhu[2], and Shuoying Chen[3]

[1] Beijing Institute of Technology,
yuyx@coretek.com.cn
[2] CoreTek System, Inc.,
zhumy@coretek.com.cn
[3] Beijing Institute of Technology,
chensy@bit.edu.cn

Abstract. This paper presents a new approach to develop device drivers for embedded system: **Devil+** language [3, 4, 7] which can automatically generate device driver code [9]. It demonstrates the whole process to develop embedded system device driver with this new language. The example project selects SHARP KEV79520 as hardware platform [8], **DeltaOS** as software platform [5, 6, 7]. Development processes are illustrated with source code. Some programming details are also explained in this paper. Finally we illustrate the benefit of the new methodology in embedded system development with the comparison between the **Devil+** approach and the traditional raw C language method.

1 Introduction

Rapid advancement of the semiconductor industry has brought equally rapid changes in the number, diversity and most notable complexity of embedded systems. There is a significant design gap in embedded system engineering tools.

This lack of automation in the hardware/software integration layer, combined with the growth of I/O complexity and design cycles reduction, is rapidly becoming a crisis. Today, the BSP task becomes an expensive bottleneck. Without automation, it will soon be impossible to meet development windows.

Fortunately, a new methodology has appeared in software domain – **Devil+** language. **Devil+** language is derived from **Devil** language [3, 4, 5], which is an extension of **Devil** language. It's a domain-specific language, which is designed specially for describing the I/O device. For its clear and simple syntax, the development of the device driver becomes more efficient. The code generated by **Devil+** compiler is C language code, which is much easier to be maintained than the code written in raw C language. And the code written in **Devil+** language is more logical and concrete than the raw C language. **Devil+** code leads the development more efficient and correct.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 418–422, 2005.

2 Hardware and Software Platforms

The device driver is the lowest level component in software. Developing device driver involves the communication with the hardware and operating system. We select SHARP KEV 79520, which is an evaluation board made by SHARP. There are plenty of devices on this board. We choose **DeltaOS** to run on this evaluation board. **DeltaOS** is a hard real-time operating system with microkernel architecture.

3 General Descriptions

The timer is so common that every computer system has it. During embedded software development, you must write timer driver [1] to let your operating system run on the board. So we choose it as an example.

There are four timers on KEV79520. Timer0 and Timer1 comprise one module and receive one common clock from the LH79520 (CPU ARM7 [1]) RCPC functional block [8]. Timer2 and Timer3 comprise a second module and receive a second which is similar to clock. Each Timer can be programmed to generate an interrupt output to the LH79520 VIC [8], and the output of Timer3 is available externally. Timers are integrated into the LH79520 SOC [8]. The four Timers can be used individually, or the Timers can be cascaded to provide longer countdown periods than would be provided by an individual Timer. When the Timers are cascaded, only the last Timer in the cascaded chain can cause an interrupt. Each Timer includes three programmable registers (Control, Load, and Clear), and one read-only register (Value).These four timers are almost the same except that they have different base address. Below in this section is the base address of every timer and hardware description of some registers in one timer.

Table 1. Timer Register Base Address

REGISTER	BASE ADDRESS
Timer0Base	0xFFFC4000
Timer1Base	0xFFFC4020
Timer2Base	0xFFFC5000
Timer3Base	0xFFFC5020

4 Using Devil+ to Write Device Drivers

4.1 Reuse Devil+ Library Code

When writing device driver, if there is **Devil+** description in **Devil+** library, we don't need to write any more codes. We can reuse this component. Now suppose

Table 2. Load Register Bit-Fields

BITS	FIELD NAME	FUNCTION
31:16		Reserved. Write '0'. Value returned on a read will be unpredictable.
15:0	LOADVAL	Load Value. A 16-bit value. Reset =0.

Table 3. Value Register Bit-Fields

BITS	FIELD NAME	FUNCTION
31:16		Reserved. Value returned on a read will be unpredictable.
15:0	CURRENTCOUNT	The current value of the Timer. Reset = 0xFFFF.

that we don't have a reuse unit in **Devil+** library when writing the timer driver of KEV79520.

4.2 Device Specification in Devil+

The first step is to describe the chip using **Devil+** language. The complete specification of timer is given as follows. Translating the natural language specification into the **Devil+** specification is rather straightforward.

```
device timer(base : bit[32] port@{0..0x102c}) {
  // Load Register
  register load(i : int{0..3}) =
    base@(-1344*i*i*i + 6048*i*i - 4672*i) : bit[32];
  unused bit load(0, 1, 2, 3)[31..16];
  segment timer_loadval(i : int{0..3}) = load(i)[15..0] : int(16);
  // Value Register
  register value(i : int{0..3}) = read base @
  (-1344*i*i*i + 6048*i*i - 4672*i + 4) : bit[32];
  unused bit value0(0, 1, 2, 3)[31..16];
  segment timer_currentcount(i : int{0..3}) = value(i)[15..0],
  volatile : int(16);
  .....
}
```

The register declaration corresponds to the hardware register. But in some hardware, the register can be multiplexed. In this circumstance we can declare different registers upon the same hardware register. The segment is a unit that programmer can reference in **Devil+** driver. It represents the special bits of the register. In **Devil+** driver we can reference segment by its name, using it like a variable in C language.

The technology used here is register and segment parameterization. Here we have four register declarations, but every one represents four timers' same function registers. The parameter expression "`-1344*i*i*i + 6048*i*i - 4672*i`"

is calculated from the four timers' base addresses. The reference to the every timer's same function register is accomplished by the parameter, which is the timer ID number. The theory also applies to segment parameterization. Many same devices have simple relationship among their memory addresses. In these situations, it's more suitable to use this technology. The characteristics of **Devil+** language make sure the device driver works correctly. The description language is simpler, clearer, and safer than C language.

4.3 Driver in Devil+

The driver written in **Devil+** is simplified by encapsulating the low-level hardware accesses and assignment in C code generated from **Devil+** specification. Here is a fragment of the code of KEV79520 timer controller device driver.

```
operation TIMER_RESET(timer_num : int(32)) : void {
  timer_loadval(timer_num)  = 0; timer_enable(timer_num)  = 0;
  timer_mode(timer_num)     = 0; timer_cascade(timer_num) = 0;
  timer_prescale(timer_num) = 0; timer_clr(timer_num)     = 0;
}
```

Because of the length of the paper is limited, just one part of the code is illustrated here. The operations' names are in uppercase. In operation part, there are some assignments to the segments, also some logical control sentences, which are very similar to C language. There are no timer hardware details can be seen in the operations.

5 Target C Code Generated from Devil+

Here is the code fragment generated by the **DeltaDevil+** compiler [7]. The static inline function is generated from the Device Specification in **Devil+**. `Drvio_ LongRegSet` is a set of macros defined in **DeltaOS** to access registers. If you use other OS, we can define these two macros to appropriate basic operations.

```
static inline void  devil_reg_set_load(u32 i, u32 v) {
  Drvio_LongRegSet(base + (-1344*i*i*i+6048*i*i-4672*i), v);
}

static inline void  devil_set_timer_loadval(u32 i, u16 v) {
  u32 temp0;
  temp0 = 0x0 | (v & 0xffff) >> 0 << 0;
  devil_reg_set_load(i, temp0);
}
```

Here is a C function generated from **Devil+** driver. The function uses the static inline functions generated from Device Specification in **Devil+** to accomplish logical operation.

```
void TIMER_RESET(int32 timer_num) {
  devil_set_timer_loadval(timer_num, 0);
  devil_set_timer_enable(timer_num, 0);
  devil_set_timer_mode(timer_num, 0);
  devil_set_timer_cascade(timer_num, 0);
  devil_set_timer_prescale(timer_num, 0);
  devil_set_timer_clr(timer_num, 0);
}
```

KEV79520 has many other devices, like LCD, touch screen, interrupt controller, Ethernet card, sound card, watch dog, serial port, IrDA, CF interface etc. **Devil+** language is used here to describe all these devices. These descriptions should be loaded to the **Devil+** library for later use. It is possible to reuse the device driver code.

6 Conclusions

The approach of **Devil+** provide a new alternative to write drivers by yourself, writing once and running everywhere. By reusing what you and others have created, it will reduce development effort and allow you to get your products to market early.

Driver programmers can throw the traditional laborious work to the **Devil+** compiler. Now the device driver development method remains on the manual level. But in the future, with the improvement of the **Devil+** compiler technology and large and complex demands of the application, using **Devil+** language to automatically generate driver codes will replace the manual method.

References

1. A.Rubini: Linux Device Drivers. O'Reilly, first edition, February 1998
2. B.N. Bershad, T.E. Anderson, E.D. Lazowska, and H.M. Levy: Lightweight remote procedure call. ACM Transactionson Computer Systems, February 1990.
3. Fabrice Mérillon Laurent Réveillère* Charles Consel* Renaud Marlet†Gilles Muller: Devil: An IDL for Hardware Programming, In OSDI 2000, pages 17-30, San Diego, October 2000.
4. Laurent Réveillère F. Mérillon, C. Consel, R. Marlet, and G. Muller: The Devil Language release 0.4. August 24, 2000
5. Lei Luo, Ming-Yuan Zhu, Qing-Li Zhang: A formal semantic definition of DEVIL. SIGPLAN Notices 38(4): 47-56 (2003)
6. M.-Y. Zhu, Lei Luo, and Guang-Ze Xiong: High-availability in δ-CORE: A formal derivation. Dedicated Systems Magazine, July 2001.
7. Qing-Li Zhang, Ming-Yuan Zhu, Shuo-Ying Chen: Automatic generation of device drivers. SIGPLAN Notices 38(6): 60-69 (2003)
8. SHARP Inc. LH79520 USER'S GUIDE
9. Tetsuro Katayama, Keizo Saisho, and Akira Fukuda: Prototype of the device driver generation system for unix-like operation systems. In proceddings of ISPSE 2000, November 2000.

On Generalizing Interrupt Handling into a Flexible Binding Model for Kernel Components

Qiming Teng[1], Xiangqun Chen[1], and Xia Zhao[1,2]

[1] Peking University, Beijing, PRC, 100871
{tqm, cherry, zhaoxia}@os.pku.edu.cn
http://os.pku.edu.cn/
[2] Beijing Business and Technical University, Beijing, PRC, 100871

Abstract. This paper presents a flexible binding implementation in JBEOS, a component based embedded operating system targeted at resource-constrained devices. This binding model features an extensible framework, which consists of two stub segments named prolog and epilog respectively. By making the binding service an ingredient of the run-time infrastructure, dynamic loading and binding of system components is supported in JBEOS. Synchronization, mutual exclusion issues are made transparent by manipulating the inter-component communications using the binding model given.

1 Introduction

To facilitate the increasing demand for extensible platforms in embedded domains, various prototypes and solutions have been proposed during the past decade, μChoices[1], EPOS[2], DEIMOS[3] are examples of such systems. These systems are all aimed at providing an extensible operating system for embedded applications.

In this paper, we first present the architecture of the JBEOS operating system and its design philosophies. Then the flexible binding model is discussed in detail as a generalization from the practice of handling interrupts by incorporating user supplied ISRs (Interrupt Service Routine). Then, we present initial lessons learnt from experiments on the component service built upon the supposed model. At last, conclusions and future works planned are given.

2 The JBEOS Nanokernel

The JBEOS is an extensible operating system designed for embedded devices. The design of JBEOS borrowed its ideas from both the XOK [4] and the DEIMOS [3] project. The primary responsibility of the nanokernel is to ensure the secure interactions among components and secure access to hardware resources, but not to provide traditional services or abstractions like threading, memory management, IPC etc. The bare image consists of only an HAL (Hardware Abstraction Layer), a component service and some low-level drivers, and thus forms the so

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 423–429, 2005.

called nanokernel in this paper. System services provided by a traditional kernel are encapsulated into dynamically loadable system components. Protections between components are dealt with the capability subsystem in the component service. Nearly all portions of the nanokernel are portable with two exceptions: the HAL is a source level component that is platform dependent, and the binding module is semi-portable due to machine-level code relocation implementations, see Fig 1.

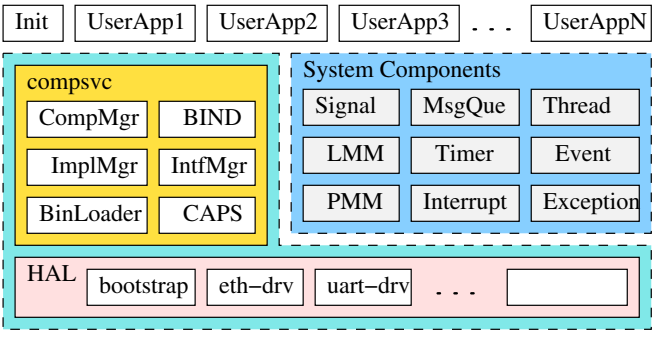


Fig. 1. The architecture of the JBEOS. Shaded boxes are parts of the *nanokernel*, while traditional system services are provided by components running from outside of the kernel. *LMM* is a component for linear memory management, while *PMM* is the physical memory management unit

Fig 1 depicts the architectural structure of the JBEOS system. The component service (`compsvc`) module consists of six major components. The `CompMgr` component is responsible for component management, i.e. component registration, de-registration, query by name or ID, and maintenance of the associations among components and implementations. The `IntfMgr` component is responsible for interface management, including interface registration/deregistration, enumerating interfaces by name and associating interfaces to implementations. The `ImplMgr` does a similar work as the `IntfMgr` component, though the entities being managed are implementations, rather than interfaces. A component in JBEOS is a binary file translated directly from a legacy ELF object file [5]. The `CAPS` component is a vital part of the `compsvc`, for its responsibility in ensuring the secure bindings among service and application components. By ejecting almost all operating system abstractions into loadable components [6], JBEOS is able to adapt to different requirements raised in embedded application domains.

The `BinLoader` entity is used to load binary reusable component files, and parsing the header of the component for `IntfMgr` and `ImplMgr`. The `BIND` module implements a specific binding model, which is detailed in the next section.

3 The Binding Model

Since one of the design goals of JBEOS is a transparent programming paradigm for the developers, so as not to bring in the burden of meeting yet another derivative of COM as done in DEIMOS [3], we have to hide the `CoInitialize` or `OleInialize` [7] and other routines from the programmers. On one hand, this transparency made it possible to optimize inter-component communications when necessary. On the other hand, reusing legacy software is easier even when one has no access to the source. An approach to hook user supplied device drivers is given in the next subsection, following which the flexible binding model is introduced.

3.1 An ISR Hooking Model

For embedded operating systems, there are cases where special hardware interfaces are used. It is nontrivial to enable applications to handle IRQs from such devices directly, because interrupts can not vector directly to C functions in user programs. A common way to do this is shown in Fig 2[8].

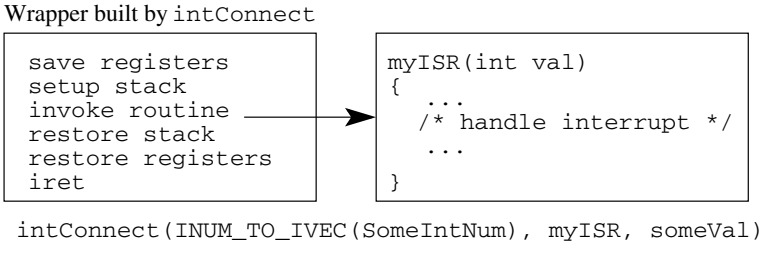


Fig. 2. Routine built by `intConnect` in embedded operating systems. By wrapping the `myISR` function, registers and stack pointers are automatically saved/restored when interrupts occur

Bindings among modules in traditional operating systems are usually done statically. An entry in the caller is fixed by the link editor when the target image is built, although a further step towards dynamic linking can be made using specific information in object files. For example, GOT (Global Offset Table) and PLT (Procedure Linking Table) sections in ELF (Executable and Linking Format) [9] are employed in UNIX-family operating systems [10][11]. We extended this linking procedure by intercepting procedure calls among system modules, i.e. two skeleton code snippets are inserted before and after the procedure call respectively. Thus, a new model for binding components was devised, which is given in the next subsection.

3.2 The Flexible Binding Model

As a component based operating system, JBEOS emphasizes the separation between interfaces and implementations in contrast to a module based OS, which further facilitates the separation of policies and mechanisms provided by the system. These features are made possible using the flexible binding model, as shown in Fig.3.

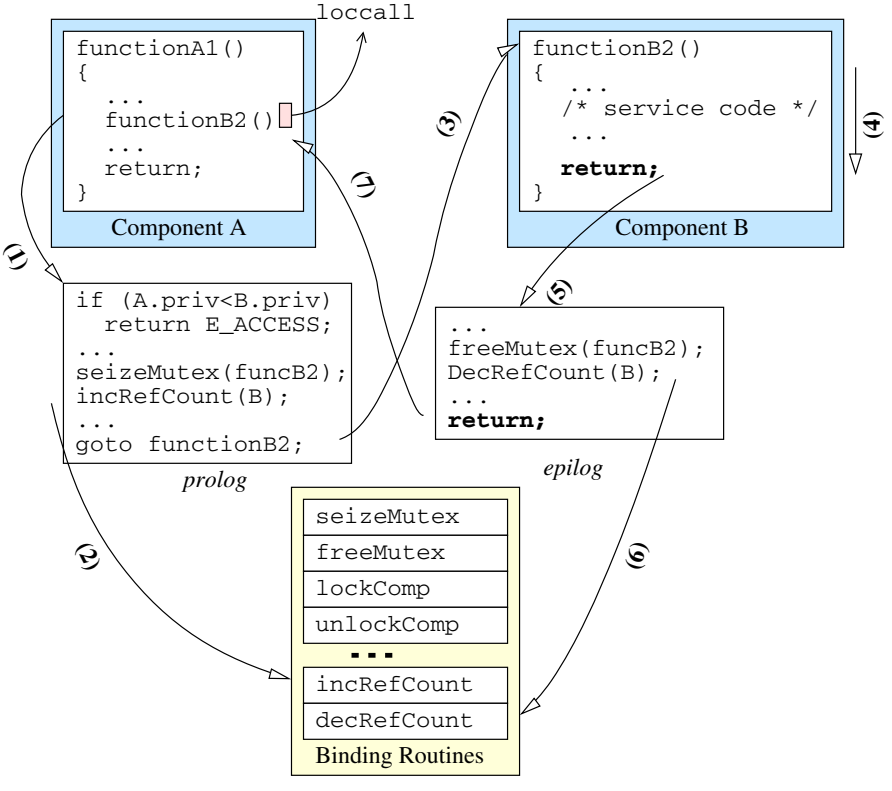


Fig. 3. The binding model illurstated. Binding routines are pre-translated binary relocatable code snippets shipped along with the `compsvc` component. This repository is customizable when building the target image

Assume that we have two components A and B to be bound, where `functionA1` in A requires a service `functionB2` from component B. This requirement is detected when component A is loaded by the `BinLoader` module of `compsvc`. After dealing with the internal relocations, `BinLoader` asks the `BIND` module to bind A with other existent components. Binary versions of *prolog* (and *epilog*) code snippet are then dynamically generated according to the interface requirement specified in the `Interface` section of A. In this case, `functionB2`

in component B claimed to be accessed exclusively. The corresponding *prolog* and *epilog* thus incorporate textttseizeMutex and `freeMutex` logic copied from the *Binding Routines* repository. The *BIND* module then redirects function call (a `REL32` entry) in the caller's code to the *prolog*, points the last `goto` instruction to `functionB2`. The *prolog* replaces the caller's address (`loccall` in fig3) by the start address of the *epilog*, so that it appears to B that `functionB2` was called by the *epilog*. When component B finishes servicing A, the `return` statement actually transfer the execution to the *epilog*, which then mimic a stack frame before return by pushing the originally saved `loccall` on A's stack. Thus component A is ignorant about the intercepting procedure happened, i.e. it seems to A the function call behaves just like other usual cases; everything is done as expected.

By following the directed lines shown in Fig.3, we can see what happens behind the scene when neither A or B is aware of the actual process undertaken. A more interesting point of this binding model is that reference counting, component locking, and cross-domain interactions can be implemented in the same way.

Prologs and epilogs are generated from binary code templates. A sample prolog is shown as following:

```
byte prologCode[] = {
 /* Three 'push $br_parameters' instructions are used for passing
  * parameters to connection code, which can be safely ignored
  */
  0x68,0x00,0x00,0x00,0x00,
  0x68,0x00,0x00,0x00,0x00,
  0x68,0x00,0x00,0x00,0x00,
  0xe8,0x00,0x00, 0x00, 0x00, /* call routinesEntry */
  0x83,0xc4,0x0c,             /* addl $12,%esp */
  0xb8,0x00,0x00,0x00, 0x00,  /* movl $loccall, %eax*/
  0x8b,0x54,0x24,0x08,        /* movl 0x8(%esp,1),%edx*/
  0x89,0x10,                  /* movl %edx, (%eax)  */
  0xba,0x00,0x00,0x00,0x00,   /* movl $_epilog, %edx*/
  0x89,0x54,0x24,0x08,        /* mov %edx,0x8(%esp,1)*/
  0xe9,0x00,0x00,0x00,0x00,   /* jmp _target_func */
  /* Target function executes here, establishing its own stack
   * frame, executing its functional logic using the parameters
   * on stack. Finally, it returns by a 'ret' instruction, which
   * is caught by the _postStub.
   */
};
```

The following is a sample epilog:

```
  byte epilogCode[] = {
    0x68,0x00,0x00,0x00,0x00,   /* pushl $br_parameter1*/
    0x68,0x00,0x00,0x00,0x00,   /* pushl $br_parameter2 */
```

```
    0x68,0x00,0x00,0x00,0x00,   /* pushl $br_parameter3 */
    0xe8,0x00,0x00,0x00,0x00,   /* call _apxStub */
    0x83,0xc4,0x0c,             /* addl $12, %esp */
    0x68,0x00,0x00,0x00,0x00,   /* pushl callerAddr + 4 */
    0xc3                        /* ret */
};
```

Both prolog and epilog code here are targeted at the Intel32 architecture, and this is why the `BIND` component is semi-portable. Zeroed bytes standing for those relocation entries are filled when binding is carried out.

4 Initial Lessons

Although adopting component based software development paradigm into embedded operating systems is deemed a promising way for extensibility and reusability requirements in practices, proposed solutions often impose a rigid programming model on developers. Systems like SPIN [12] restrict the extensions to be programmed in Modula-3, a type-safe language. MMLite [13] and DEIMOS [3] adopt variants of COM [14] as their component models, forcing developers and their programs to be aware of the underlying component run-time facilities. The binding model proposed in this paper is a transparent one to developers. However, compiled objects should be converted into an EBC format in order to reinforce the interface specification in the component deployed.

Another aspect of this binding model is its capability to implement zero-cost interactions among components. By filling the address of the method in the target component, the BIND component can simulate what a link editor does. On the other hand, remote bindings in a networking environment is possible by providing marshalling functions in the Binding Routines repository, although we have not yet implement such a scheme. All performance costs in the binding model are inherent ones in the interacting scheme, because no additional costs are introduced except for some stack manipulations where additional memory accesses are required.

The safety of bindings are ensured by the CAPS unit in the BIND module, which centralize protection issues like access control, as EROS[15] has done.

5 Conclusions and Future Work

Flexible, secure bindings in a component based operating system have notable impacts in the customizability, performance, and dependability of the target system. The binding model presented in this paper generalized an interrupt handling model into a flexible framework, where prolog and epilog code snippets are inserted as the wrapper of methods from target service component. Future works are to be carried out with respect to an optimal footprint, reimplementation of traditional IPCs.

Acknowlegements

Thanks to Xinjie HU, Wei ZHU and Jianjun HU who took an active part in developing the prototype implementations.

References

1. Campbell, R., Tan S. M.: *μChoices: An Object-Oriented Multimedia Operating System.* In Proceedings of the Fifth Workshop on Hot Topics in Operating Systems, Orcas Island, WA, May 1995. New York: ACM Press, 1995, pp. 90-94.
2. Fröhlich A., *Application-Oriented Operating Systems.* Ph.D Thesis, Germany: GMD-Forschungszentrum Informationstechnik GmbH, 2001.
3. Clark M., *Operating System Support for Emerging Application Domains.* Ph.D Thesis, Lancaster, UK: Lancaster University Computing Department, 2000.
4. Engler D. R., *The Exokernel Operating System Architecture.* PhD Thesis, Massachusetts Institute of Technology, October 1998.
5. TENG Qiming, CHEN Xiangqun, ZHAO Xia, *On Building Reusable EOS Components from ELF Object Files.* Journal of Software, 2004, 15(sup)
6. Engler D., Kaashoek M. F., *Exterminate All Operating System Abstractions.* In Proceedings of the Fifth Workshop on Hot Topics in Operating Systems, May 1995. pp. 78-85.
7. Eddon G., Eddon H., Inside Microsoft COM+ Base Services, Microsoft Press, 1999.
8. WindRiver Systems, VxWorks Programmer's Guide, 2000.
9. TIS(Tool Interface Standard) Committee, *Executable and Linking Format (ELF) Specification*, Version 1.2, May 1995.
10. Levine J. R., *Linkers and Loaders*, Morgan Kaufmann, 2000.
11. Rubini A., Corbet J., *Linux Device Drivers*, 2nd Edition. OReilly, June 2001.
12. Bershad B., Savage S., Pardyak P., et al. *Extensibility, Safety and Performance in the SPIN OperatingSystem.* In Proceedings of the 15th Symposium on Operating Systems Principles, NewCopper Mountain, CO, Dec 1995, NY: ACM Press, 1995. pp.267-284.
13. Helander J., Forin A., *MMLite: A Highly Componentized System Architecture.* In Proceedings of the 8th ACM SIOPGS European Workshop, pp.96-103, Sintra Portugal, September 1998.
14. Microsoft Corp., *The Component Object Model Specification*, Version 0.9, 1995.
15. Shapiro J. S., Hardy N., *EROS: A Principle-Driven Operating System from the Ground Up.* IEEE Software, 19(1):26-33, 2002.

Research Directions for Embedded Operating Systems

Xiangqun Chen[1], Xia Zhao[1,2], and Qiming Teng[1]

[1] Operating System Laboratory, Institute of EECS, Peking Univ., Beijing, 86-100871
{cherry, zhaox, tqm}@cs.pku.edu.cn
http://os.pku.edu.cn
[2] School of Computer Science, Beijing Technology and Business Univ., Beijing, 86-10037
zhaox@th.bpbu.edu.cn
http://www.bpbu.edu.cn

Abstract. A brief survey for recent research works on embedded operating systems (EOSs) is presented, including component based EOSs, energy-aware EOSs, secure EOSs, and EOSs for sensor networks.

1 Introduction

Embedded operating systems are one of the enabling technologies for areas such as industrial process control, precision agriculture, defense systems, and consumer electronics etc. It serves an important role in the post-PC era. In fact, modern embedded systems are becoming increasingly important elements of the infrastructure for the ecosystem of the world. New research directions are emerging while embedded systems are becoming more ubiquitous and pervasive.

This paper presents a survey on research directions of modern embedded operating systems. Directions surveyed in this paper include: composibility, energy-efficiency, dependability, performance guarantees of embedded operating systems, and systems for emerging application domains , eg. sensor networks.

2 Composibility

Embedded applications can be deployed in many fields, each of which has its unique set of requirements. Requirements with respect to processor performance, memory, energy, cost, operation mode, and time to market vary from one field to another, from time to time. A key challenge is to provide an embedded operating system with a high degree of tailorability, which can evolve and make use of legacy components. Composibility has long been a key issue for these systems.

A promising solution to this challenge is to introduce component based design to EOSs. There are some industrial component based embedded operating systems and commercial ones available. Industrial products examples include: MMLite [1], Pebble [2] and etc. Some academic systems are examined briefly in this section, namely OSKit [3], PURE [4], 2K [5], and JBEOS [6].

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 430–436, 2005.

2.1 OSKit

The OSKit [3] system developed at Utah university is a toolkit for system research project. OSKit offers a component library for operating system configuration and customization. The authors of OSKit made great efforts to minimize the number of interactions and dependencies among components. This effort results in enhanced flexibility between components and code created independently by other developers. But OSKit does not provide any guidelines to build an operating system out of these components. Another issue of OSkit is that it does not focus on embedded systems.

2.2 PURE

PURE [4] is an object-oriented operating system focused on deeply embedded systems, with extreme resource constraints in terms of memory, CPU, and power consumption. The smallest building unit of PURE is a class. For configuration purposes, PURE provides tools to specify user requirements for the target system. An annotation language was developed to provide necessary information such as dependency and attributes, so that the composition tool can evaluate and choose the right building-units.

2.3 2K

The 2K [5] system is a reflective, component-based operating system whose main goal is to provide a generic framework that supports adaptation in a network-centric computing environment. A component in 2K is a dynamically loadable unit residing in a dynamic link library (DLL). Components can be loaded or unloaded according to user's needs. No analysis tool is provided to determine correctness and performance of 2K. However, components can decide if adaptation is required based on the observed system state. The system provides configuration and reconfiguration capabilities based on static and dynamic dependencies.

2.4 JBEOS

JBEOS is a component-based embedded operating system developed at EECS of Peking University. A component library for EOS composition was constructed. The reusable EOS components are directly produced out of complied objects, i.e. ELF object files [6]. The JBEOS component model has three layers: the platform, the kernel and the feature layer. An XML-based component description language, named XCODE (eXtensible COmponent DEscription language), is defined with some advantages: independence from tools, platforms and components, exchangeable format, extensible, and eligible for varying scenarios. The experiment result shows that XCODE is suitable for component based EOS development [7].

The current research objective is to develop a model for component composition that encompasses domains such as function, time, and fault tolerance. Further research should concentrate on issues related to adaptive high-performance fault-tolerant embedded systems that can satisfy functional and extra-functional requirements. These systems should exhibit a priori correctness and acceptable performance while ensuring graceful degradation in presence of sporadic failures. Unexpected requirements for functions and performance can be satisfied by employing some scheme that makes the system evolve in a predictable way.

3 Energy Efficiency

Many embedded systems have energy constrains. Traditional operating systems gave little consideration on energy consumption, if any. Nowadays, a new resource management perspective should take on energy management as an aspect of the resource management functionality of an operating system. Dynamic power management as a promising research direction is discussed first in this section, followed by a summary of research status on energy effects of an OS.

3.1 Dynamic Power Management (DPM)

At the operating system level, basic dynamic power management issues being considered include: Dynamic Voltage Scaling (DVS) [8][9] for the processor unit, and energy saving techniques for the I/O devices [10][11].

Software-conducted power management has gained much attention from both research communities and industry vendors. The first standard developed is the *Advanced Power Management (APM)* specification, a BIOS-based power management specification for PC platform. Due to problems found in APM, the *Advanced Configuration and Power Interface (ACPI)* specification was developed. Power management related decisions are made now by the operating system rather than the BIOS firmware. ACPI establishes an industry-standard interface for OS-directed configuration and power management on different platforms [12]. However, embedded platforms tend to exhibit distinguishable peculiarities that make ACPI not directly applicable. A generic adaptation of the ACPI is anticipated by embedded developers.

A research group from EECS, Peking University, has developed two DVS algorithms [13] for dynamic power management. The preliminary experiments have achieved significant energy saving while simultaneously preserving timeliness guarantees the requirements by real-time tasks. The measurements indicate that 15% to 35% energy savings can be achieved on the Samsung S3C44B0X platform running the DeltaOS [14] real-time embedded operating system.

3.2 Engery Effects of the OS Software

An operating system, as the resource manager of a computer system, should take in energy as a special kind of consumable resources. However, operating

systems, like application softwares, should be assumed as one of the major energy consumers at the same time. An operating system has significant impacts on the energy efficiency of the embedded system. Researches on adapting software to manage energy consumption include [15][16][17].

As pointed out by Vahdat et al. [18], a systematic reexamination of all aspects of operating system design and implementation from the energy efficiency perspective is necessary to determine or to predict power consumption of the system software. Services, policies, mechanisms, and the internal structure of an operating system are all factors of energy efficiency.

According to research results published by T. K. Tan et al. [19], a systematic framework for software architecture transformations to reduce energy consumption is attainable. The experiment result presented is quite impressive: up to 66.1% reduction in energy can be obtained. During the early stage in the development of embedded software and operating systems, energy should be taken into consideration seriously. Further research works are encouraged in the following areas: hardware/software co-design to reduce power consumption, off-line prediction of power consumption, techniques to conserve energy while preserving the real-time performance requirements of the target system.

4 Dependability

Computer system has never achieved a high trustworthiness when compared to hydroelectricity supply despite the latter can cause catastrophic accidents to human. When embedded applications are spreading rapidly into nearly every corner of our workplaces and daily lives, the trustworthiness of an embedded system has become one of the focus areas of research recently. The term *Trustworthiness* is in a sense the macroscopical description of the dependability of a system. Laprie [20] refers to dependability as the trustworthiness of a computer system, and defines it in terms of four attributes: *reliability*, *availability*, *safety* and *security*. *Reliability* is the aspect of dependability referring to the continuity of a system's correct service; *Availability* refers to a system's readiness for usage; *Safety* concerns itself with systems avoiding catastrophic consequences on their environment and/or operators; *Security* captures the ability of a system to prevent unauthorized access to information. Due to the space limitation, only security for EOSs is investigated here.

A well-known standard is the *Common Criteria* specification by CCIE [21] which defines 7 levels of security a product provides, named as *Evaluated Assurance Levels (EALs)*. The basic requirements for an OS to be secure are [22]: Non-bypassable, Evaluatable, Always invoked, and Tamper-proof. These four basic tenets are very hard to achieve simultaneously. A viable approach to build a secure EOS, among others, is to construct a separation kernel, which is strictly limited in functions provided. Given the small code base of the kernel resulted from removing as much work out of the kernel space into application space, the security of the kernel can be formally proven. A kernel in this case degenerates into a channel enforcing isolations among applications. The security policies are thus always invoked and are non-bypassable.

One example is the *DO-178B* standard used by the FAA for flight systems [23] certification purposes. Comparison between *DO-178B* and *Common Criteria* reveals that software that conforms to the DO-178B Level A criteria will map closely to either EAL 4 or 5 with some additional works [24]. Examples of embedded operating system satisfying the DO-178B Level A standard are emerging, e.g. LynxOS-178, whose core is LynxOS, a hard UNIX-style RTOS. Further researches are encouraged with certifiable dependability and QoS assurance in the presence of real-time.

5 EOSs for Emerging Application Domains

In a rapidly developing world, pervasive computing is apparently a wheel in the family of next-generation computer systems. Emerging domains, most of which may be unknown, may raise unexpected requirements to the EOS on which applications run. In this section, EOS for sensor networks is taken as an example to illuminate some research issues when developing application-specific EOSs for emerging domains.

The emergence of sensor networks is a natural evolution of the trend in computing and communication technologies toward smaller, more powerful information appliances that are becoming ubiquitous [25]. Key research areas for this flavor of EOSs are: self-configuration and adaptive coordination, trustworthiness, and computational models. TinyOS developed at Berkeley is an example of such systems [26]. TinyOS provides an extremely efficient multi-threading engine with a two-level scheduling structure, so that processing overhead to hardware events can be reduced to a minimal degree. High concurrency is handled using an event model with small space cost.

Another research issue is the simulation of sensor networks. These simulation environments have to scale well enough to handle up to thousands of nodes at a high fidelity. A simulator that can accurately capture the dynamic properties of a sensor network is favorable. Sample simulators are emerging, e.g. TOSSIM [27], Ns-2 [28].

Each application domain has its unique set of functional and extra-functional requirements, so the research issues have close relationships to evolvability of an EOS. An open infrastructure can make an embedded system more evolvable than ever: hardware and software components insertion and removal can be made in a safe and cost-effective way.

6 Conclusions

This paper discussed several strategic research directions for embedded operating systems. Researches related to composability are more active than in the other fields. Current research related to energy efficiency, dependability and meeting requirements from emerging domains are preliminary. There is still a long way to go for academic community and industrial vendors, so a closer cooperation between academic researchers and industrial organizations is anticipated.

Acknowlegements

This work was supported by the *National High Technology Research and Development Program of China* (863 Program) under Grant No.2002AA1Z2204, and the *National Natural Science Foundation of China* under Grant No. 60373001.

References

1. Helander J., Forin A.: MMLite: A Highly Componentized System Architecture. In: Proc. of 8th ACM Special Interest Group on Operating Systems European Workshop (SIGOPS). ACM Press, New York. (1998) 96–103
2. Gabber E., Bruno J., Brustoloni J., Silberschatz A., Small C.: The Pebble Component-Based Operating System. In: Proc. USENIX Annual Technical Conference. USENIX Assoc, California. (1999) 267–282
3. Ford B., Back G., Benson G., Lepreau J., Lin A., Shivers O.: The Flux OSKit: A Substrate for Kernel and Language Research. In: Proc. of the 16th ACM Symposium on Operating System Principles. ACM Press. (1997) 38–51
4. Beuche D., Guerrouat A., Papajewski H., Schroder-Preikschat W., Spinczyk O., Spinczyk U.: The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems. In: Proc. of 2nd IEEE International Symp. on Object Oriented Real-Time Distributed Computing. (1999)
5. Kon F., Singhai A., Campbell R.H., Carvalho D., et al.: 2K: A Reflective, Component-Based Operating System for Rapidly Changing Environments. In: ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems. (1998)
6. Teng Q., Chen X.: On Building Reuseable EOS Components from ELF Object Files. Journal of Software, Vol. 15 (2004) 157–163
7. Teng Q., Chen X.: XCODE: A Extensible Component Description Language for System Software. Journal (Natural Sciences) Of Peking University. Vol. 41 (2004) 388–396
8. Pering T., Brodersen R.: The Simulation and Evaluation of Dynamic Voltage Scaling Algo-rithms. In: Proc. of the International Symp. on Low-Power Electronics and Design. (1998) 76–81
9. Flautner K., Reinhardt S., Mudge T.: Automatic performance-setting for dynamic voltage scaling. In: Proc. of the 7th Conference on Mobile Computing and Networking. (2001)
10. Swaminathan V., Chakrabarty K., Iyengar S. S.: Dynamic I/O Power Management for Hard Real-Time Systems. International Symposium on Hardware/Software Co-Design. (2001) 237–242
11. Benini L., Bogliolo A., Micheli G. D.: A Survey of Design Techniques for System-level Dy-namic Power Management. IEEE Transactions on VLSI Systems, Vol.8(3) (2000)
12. The ACPI Standard, available at http://www.acpi.info/
13. Zhao X., Chen X., Gao Z., et al. Dynamic Voltage Scaling for Power-Aware Embedded Operating System. In Proc of 2004 Intl. Conf. on Embedded System and Software. (2004)
14. DeltaOS: http://www.coretek.com.cn/
15. Frakas K., Flinn J., Back G., Grunwald D., Anderson J.: Quantifying the Energy Consumption of a Pocket Computer and a Java Virtual Machine. In Proc. of SIGMETRICS'00. (2000)

16. Flinn J., Satyanarayanan M.. Energy-aware Adaptation for Mobile Applications. In Proc. ACM SOSP. (1999) 48–63
17. Lorch J.R., Smith A.J.: Software Strategies for Portable Computer Energy Management. IEEE Personal Communications, Vol.5(3) (1998) 60–73
18. Vahdat A., Lebeck A., Ellis C.S.: Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency. In: Proc. 9th ACM SIGOPS European Workshop. (2000)
19. Tan T.K., Raghunathan A., Jha N.K.: Software Architectural Transformations: A New Approach to Low Energy Embedded Software. In: Proc. Design Automation & Test in Europe. (2003)
20. Laprie J.C.(ed.): Dependability: Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems. Springer Verlag. (1991)
21. NIST, Common Criteria for Information Security Evaluation. Parts(1-3). (1999). Available at: http://csrc.nist.gov/cc
22. Lynx. http://www.lynuxworks.com/products/whitepapers.php3
23. RTCA: DO-178B/ED-12B. Software Considerations in Airborne Systems and Equipment Certification. (1992)
24. Alves-Foss J., Taylor C., Rinker B.: Merging Safety and Assurance: The Process of Dual Certification for Software. (2002) Avai at http://www.csds.uidaho.edu/ jimaf
25. National Academies: Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers. ISBN: 0-309-07568-8. (2001)
26. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K.: System architecture directions for networked sensors. In: Proc. of ASPLOS IX. (2000)
27. Levis P., Lee N., Welsh M.,Culler D.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In: Proc. of the 1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003) (2003)
28. NS-2: http://www.isi.edu/nsnam/ns/

SmartOSEK: A Real-Time Operating System for Automotive Electronics

Minde Zhao, Zhaohui Wu, Guoqing Yang, Lei Wang, and Wei Chen

Zhejiang University Hangzhou, P. R. China, 310027
{zmdd48, wzh, ygq78, alwaysbeing}@cs.zju.edu.cn

Abstract. This paper puts forward SmartOSEK, a dependable platform for automobile electronics, which consists of an operating system compliant with OSEK/VDX specifications and an integrated development environment (IDE) that consists of many convenient tools, such as visual designer, system generator, time analyst, scheduling analyst, and running tracer. In the operating system, SmartOSEK OS and SmartOSEK COM are presented. In the IDE, we apply the graphic design, automatic code generation and time analysis to help developers devote their mind to the modeling of the applications. A design example of automated transmission system based on SmartOSEK platform is given, and good results are achieved.

1 Introduction[1]

Vast arrays of embedded technologies are used increasingly in automobiles to improve the performance and reliability of the system, and the system becomes complex. To manage the complex system, it is desirable to use operating system in the automotive electronic system. An operating system will perform task management, abstracting away task switching and synchronization from the application code, and simplifying application development and verification.

A consortium, led by automotive and microcontroller corporations, has developed the OSEK/VDX (Open systems and corresponding interfaces for automotive electronics /Vehicle Distributed executive) standard, which includes specifications for operating system, communication subsystem, and network management subsystem [1]. For simplicity OSEK will be used instead of OSEK/VDX in the paper. The goal of OSEK standard is to improve the reusability and compatibility of software. The OSEK standard allows multiple parties to develop applications and subsystems that can work together and be integrated easily.

Based on the OSEK operating system specification, we have developed a small, efficient operating system called SmartOSEK, which is implemented on the Freescale MPC555 microprocessor. An integrated development environment is developed for SmartOSEK. The hierarchy of SmartOSEK and IDE is shown in Fig. 1.

[1] This work is supported by 863 National High Technology Program under Grant No. 2003AA1Z2141.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 437-442, 2005.

The rest of this paper is organized in the following way. We present an overview of SmartOSEK platform in section 2; a development example is introduced in section 3 based on SmartOSEK platform; in section 4 we introduce time analysis technology for SmartOSEK platform. The paper is concluded in section 5.

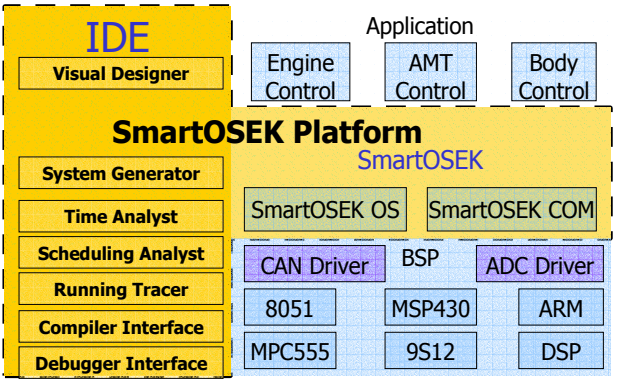


Fig. 1. The architecture of SmartOSEK platform

2 Overview of SmartOSEK Platform

SmartOSEK platform is developed by Embedded System Engineering Laboratory in Zhejiang University. It consists of an operating system compliant with OSEK specifications and an integrated development environment that consists of many convenient tools. The operating system consists of a kernel named SmartOSEK OS and a subsystem name SmartOSEK COM, and we will discuss them in detail.

2.1 SmartOSEK OS

SmartOSEK OS is a small embedded operating system kernel designed to meet the OSEK OS 2.2.1 specifications [2]. It contains the components, namely task management, scheduling, resource management, and interrupts handling. As it is designed for embedded systems that are typically running on low processing power and low-capacity systems with static applications, it does not provide the mechanisms for memory management or dynamic modification of the task set.

Task Management and Scheduling

As the task set is statically defined for the system at the time of system generation, task management is quite limited in SmartOSEK OS. A basic OSEK task can switch between active states (either READY or RUNNING) and an inactive state (SUSPENDED). As for an extended task, a WAITING state can block its execution until some event occurs. Tasks that are ready to run achieve temporal partitions on the processor. Basically the scheduling strategy uses a static priority scheme to conform

to OSEK specification. Higher priority tasks are processed before any lower priority tasks. Additionally, tasks may be preemptive or non-preemptive. Aiming at the best utilization of the processor time and the responsive time for aperiodic non-blocking tasks, a scheme of sporadic server [3] is implemented within the scheduler. This server is treated as a task that has the highest priority among all the tasks.

Synchronization

To share and protect critical data between tasks, resources in SmartOSEK OS are managed in a specific way. Only one task can hold a resource at a given time. Two underlying problems will occur when resources are used, namely priority inversion [4][5] and deadlock [6]. To avoid these problems, priority ceiling protocol [7] is implemented in SmartOSEK OS. Any task holding a resource will have priority or at least as great as any other tasks that may request the resource, so these other tasks will not run until the resource is released. So when a task tries to obtain a resource, no task can possibly be holding that resource, and hence no blocking can occur when obtaining resources.

Alarms

SmartOSEK provides alarms to facilitate some time-based task activation. An alarm is tied to a counter or system clock and is triggered when the count reaches the alarm value. When an alarm is triggered, the system will either activate a task or signal that task with an event. The alarms are defined statically at system generation, but the time at which they are triggered is set dynamically to either relative or absolute values. If a task is not related to an alarm during system generation, it will be considered an aperiodic task and will be scheduled within sporadic server.

2.2 SmartOSEK COM

SmartOSEK COM compliant with the OSEK COM specification offers services to transfer data between tasks and/or interrupt service routines. Access to SmartOSEK COM services is only possible via the specified Application Program Interface (API). SmartOSEK COM is based on messages. A message contains application-specific data. Messages and message properties are configured statically via the OSEK Implementation Language (OIL) [8].

In the case of internal communication, the Interaction Layer (IL) makes the message data immediately available to the receiver. In the case of external communication the IL packs one or more messages into assigned Interaction Layer Protocol Data Units (I-PDU) and passes them to the underlying layer. The functionality of internal communication is a sub-set of the functionality of external communication. Internal-external communication occurs when the same message is sent internally as well as externally.

The data that is communicated between the IL and the underlying layer is organized into I-PDUs which contain one or more messages. A message must occupy contiguous bits within an I-PDU and must not be split across I-PDUs. Within an I-PDU, messages are bit-aligned. The size of a message is specified in bits. The IL offers an API to handle messages. The API provides services for initialization, data

transfer and communication management. Services transmitting messages over network are non-blocking. This implies, for example, that a service that sends a message may be unable to return a final transmission status because the transfer to the network is still in progress. SmartOSEK COM provides notification mechanisms for an application to determine the status of a transmission or reception [9].

2.3 SmartOSEK IDE

Visual Designer
In SmartOSEK platform, we implement a visual designer which adopts graph based programming. It supports graphic modeling and automatic code generation, so developers can devote their minds to modeling and designing arithmetic. The visual designer provides most objects of OSEK specifications in the form of graph, such as task, alarm, event, message and APIs; developers can move them from tool bar to the designing area freely. When the control model is built, visual designer can generate code in C language automatically. So the efficiency for automobile application development is improved greatly by graph based programming.

Time Analyst
In hard real-time distributed control systems, jobs have stringent timing constraints and are often required to be executed on a sequence of processors. Timing constraints are typically given in form of end-to-end deadline. A job in such a system meets its timing constraint if it completes before its end-to-end deadline. In order to analyze the process time of system, IDE presents a time analysis tool to calculate each task in the real-time system, and it provides references for system scheduling. Developers can master the tasks' time properties by the tool.

System Generator
In the OSEK specification, much of the system is statically defined, and SmartOSEK provides system generator to configure the real-time system. The configuration of the system can be implemented by OSEK implementation language (OIL), which is specified in OSEK OIL specification. The system generator configures the system according to the OIL files or the visual designer. The visual designer can create OIL files automatically. The OIL files provide the definition of the whole system, including all objects in OSEK. In OIL files, each task is defined by its task name, priority, alarms, resources, and the amount of stack space needed and how it is scheduled Other objects are defined in the same way in OSEK.

Running Tracer
As the real-time system runs in the embedded devices, it is hard for developers to get the running state of each task. When the running system encounters failure, the developer cannot find the position of the failure. The IDE presents a running tracer for developers to trace the system running in the embedded device. Developers can get the state of the running system at any time by the tracer. In the top of the tracer, graphic display is presented, which consists of running task, ready queue, waiting

queue, suspend queue, and the objects in the system. In the bottom of the tracer, the running state of the real-time system is displayed in the text form.

3 SmartOSEK Based Development

This section demonstrates the development based on SmartOSEK platform. SmartOSEK provides a visual designer which supports automatic code generation. Fig. 2 shows the visual designer for SmartOSEK. In the left of the designer, tool bar is composed of most objects in SmartOSEK, including task, alarm, event, APIs and so on. In the right of the designer, graphic design area and code area are presented. Developers can model their application in graphic design area by moving the objects from the left to the right; they can double-click the objects to set their attributes. The designer generates the corresponding code in the code area.

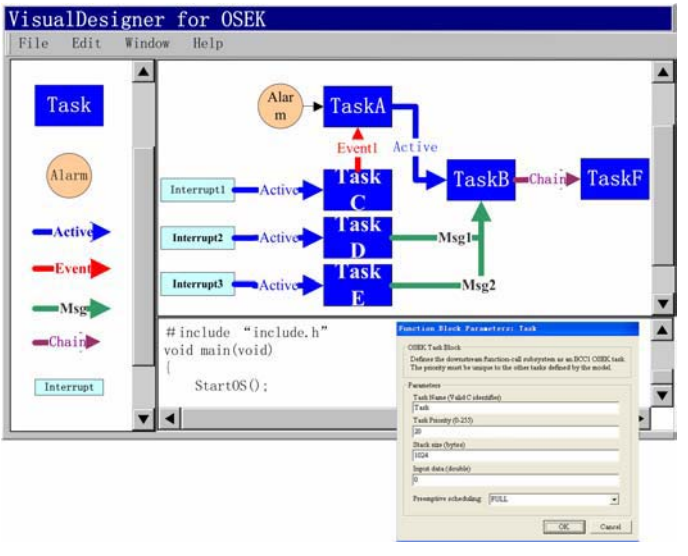


Fig. 2. Visual designer for OSEK

We take the designing of the automated transmission system as an example. Five-speed gearbox is assumed. Each of three shifting rods is actuated by servo that can shift the collar from neutral to the first gear or to the second gear. The clutch actuated by servo is supposed. Gearbox and clutch are controlled by software running in electronic transmission unit (ETU), which consists of three tasks.

In the system, TaskA, Automated Transmission Task selects appropriate gear according to the engine condition. It is assigned by the lowest priority 2 and it is periodically activated by an alarm. The speed of the engine is captured by Interrupt1, and then the ISR1 is called, which activates TaskC. If TaskC finds that it is the time to shift the gearbox, it sets an event, Event1 to TaskA. TaskA gets the event and activates TaskB, GearBox Task. TaskB opens the clutch, disengages current gear, and engages desired gear. TaskB has to read the messages of the speed of engine and the

speed of the vehicle to determine when to close clutch. Msg1 sent by TaskD presents the speed of engine, and Msg2 sent by TaskE presents the speed of vehicle. This task is assigned by the middle priority 3. When TaskB disengages or engages any gear, it specifies which shifting rod servo and in which direction it is necessary to move. Then it activates the highest priority task TaskF, Rod Servo Task which controls the movement to the desired position.

4 Conclusions and Future Work

Operating system plays an important role in automobile electronics, and in this paper we bring forward the framework of SmartOSEK platform. By the SmartOSEK platform, it is more convenient to develop automobile electronics applications. In the visual designer, we apply the graphic design and automatic code generation to help the developers devote their minds to the modeling of the applications. SmartOSEK platform also provides a time analysis tool for the developers to optimize the design. We design an automated transmission system by SmartOSEK, thus the development effort is decreased and the time is shortened.

Some tasks are left for future work. Firstly, the network management subsystem compliant with OSEK NM specification will be developed to support multi-ECUs' collaboration. Secondly, the visual designer should support high level modeling.

References

1. K. M. Zuberi, P. Pillai, K. G. Shin, W. Nagaura, T. Imai, and S. Suzuki, EMERALDS-OSEK: A Small Real-Time Operating System for Automotive Control and Monitoring, in Society of Automotive Engineers Congress and Exposition (March 1999). SAE Technical Paper Series 1999-01-1102.
2. OSEK/VDX: OSEK/VDX Operating System Specification Version 2.2.2. July 5th, 2004 Available: www.osek-vdx. org.
3. S. Ramos-Thuel, P. Lehoczky. On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems. In proc.14th IEEE Real-Time Systems Symposium, pages 160–171, North Carolina, USA, December 1993.
4. D.Locke, L.Sha, R.Rajikumar, J.Lehoczky, G.Burns. Priority inversion and its control: An experimental investigation. ACM SIGAda Ada Letters, Volume 8, Issue 7, November 1988.
5. D. Locke, L. Sha, R. Rajikumar, J. Lehoczky, G. Burns. Priority inversion and its control: An experimental investigation. Proceedings of the second international workshop on Real-time Ada issues, June 1988.
6. Dieter.Zöbel. The Deadlock problem: a classifying bibliography. ACM SIGOPS Operating Systems Review, Volume17, Issue 4,October 1983.
7. Jaehong Shim, Kyunghee Choi, Gihyun Jung, Seungkyu Park, HyeonSik Shin, Dongyoon Kim. Priority inversion handling in microkernel-based Real-Time Mike. Proceedings of the Third International Workshop on Real-Time Computing Systems Application, October 1996.
8. OSEK/VDX: OSEK/VDX System Generation OIL: OSEK Implementation Language Version 25. July 1st, 2004 Available: www.osek-vdx. org,
9. OSEK/VDX: OSEK/VDX Communication Specification Version 3.0.3. www.osek-vdx. org, July 20, 2004

A Functionality Based Instruction Level Software Power Estimation Model for Embedded RISC Processors

Jia Chen, Sheng-yuan Wang, Yuan Dong, Gui-lan Dai, Yang Yang

Dept. of Computer Sci. & Tech., Tsinghua Univ., Beijing 100084, China
chenjia02@mails.tsinghua.edu.cn

Abstract. This paper describes a functionality-based instruction-level power analysis model, which aims at reducing workload of computing inter-instruction power and keeping the convenience to observe necessary parameters from a source-code description. The model treats the total power as the sum of basic power of individual functional component and switching power of consecutive components pairs. To get the switching power, the switching activities between two functional components are treated as one changing from working state to sleeping state and the other from sleeping state to working state. NOP instructions are used to model transitions between the two states. The model is experimentally validated on a wide range of embedded software routines. Experiments show that our model is within 95% accuracy on the average, and can reduce the workload from a complexity of $O(n^2)$, which is the workload of traditional instruction-level energy estimation techniques, to a complexity of $O(n)$.

1 Introduction

Energy consumption has become one of the critical constraints in system-level design of embedded applications. Since power consumption depends significantly on the software being executed, a shift towards power estimation from the software standpoint is natural.

A well-designed model for software energy estimation should satisfy the following four qualifications: accuracy, simplicity, accountability and retargetability.

This paper presents an instruction-level functionality-based energy estimation model for embedded software, with major contributions listed as follows.

1. It is easy to identify important functional components which have significance in the energy consumption.
2. By taking a functionality-based standpoint, the model has effectively reduced the computing complexity.
3. The model building approach is generally applicable to a wide variety of processors, although experiments are carried out on MIPS like simulator.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 443-448, 2005.

2 Related Work

Initial work has been done by Tiwari [1]. He computes the energy consumption by summing up basic energy costs for individual instructions and switching power of consecutive instruction pairs. A problem is that there are too many instruction pairs, and the workload to compute switching power is onerous. To solve it, Mehta gathers those instructions with similar power into a cluster [2]. Klass proposes a model in which inter-instruction effects are measured by considering only the additional energy consumption when a generic instruction is executed after an NOP [3]. Anantha P. Chandrakasan's [8] research work shows that lots of overheads are common across instructions and the overall consumption of a program primarily depends on operating frequency and voltage. He proposes the first order model, the second order model and implements them into JouleTrack.

3 Functionality-Based Power Consumption Estimation Model

3.1 Model Description

Many components are left unused when an individual instruction is running. Since each instruction uses different components, switching activities of instructions would lead to switching of components. The execution process of a program turns into a working and switching process of functional component. Based on such a viewpoint, we construct our power estimation model as the following equation.

$$P = \sum_{i=1}^{m} \sum_{j=1}^{n} B(i, j) + \sum_{i=1}^{m} \sum_{j=1}^{n} S(i, j) + O \tag{1}$$

Suppose that there are n instructions and m functional components. P is the total power consumption; $B(i, j)$ is the basic power consumption of component i caused by instruction j; $S(i, j)$ is the switching power of component i caused by instruction j, which includes the startup stage power and stop stage power of the ports used on component i; and O is other factors that might influence the total power consumption, such as pipeline stalls.

3.2 Basic Power Consumption of Functional Components

When two instructions use the same component, the basic power consumption might be different for the following reasons.

1. For multi-ported hardware units, conditional clocking is used to disable part of a hardware unit to reduce power consumption. Different instructions may use different ports, thus the power of hardware unit might differ accordingly.
2. Different instructions might engross a certain functional component for different number of clock cycles.

3. Power consumption is actually the product of voltage and electric current. The representations of "0" and "1" in binary system are physically differed by voltage. Since the binary codes of instructions are different, the voltage also defers and so does the basic power consumption of each component.

Considering above factors, a table is used to record hardware resources used by each instruction. "0" is used if *Instruction i* occupies *Component j*. Otherwise a coefficient α_{ij} is used, which represents basic power of *Component j* when running *Instruction i*. It can be measured through the way of instruction profiling [9].

We repeat *Instruction i* for 100,000 times, with different operands (such as register names and immediate values). A cycle-by-cycle instruction-level simulator, SimpleScalar/Wattch tool set [4,7], version 3.0, is used to collect statistical data of the power dissipation of each functional component. After profiling every individual instruction, we get all the coefficients.

3.3 Switching Power of Functional Components

For each port of components, there is additional power consumption when it changes between the working state and the sleeping state. The switching power is actually the sum of the additional power consumption of these two stages.

To get this additional power, special test samples must be designed to keep the components and ports continuously changing their states. Since the NOP operation hardly wakes up any component, we construct loops by adding several NOPs between individual instructions.

Several NOPs instead of one are used for two reasons. First, enough time should be left for components to change their states. For example, for an *m-stage* pipeline, *m-1* NOPs should be added before the components completely finish their switching activities. Second, in our test programs there is high probability of hardware resource dependency. By adding *m-1* NOPs, the probability of resource dependency is lowered.

4 Experiments and Results

Our simulations are done on SimpleScalar/Wattch [4,7], version 3.0 with RTL level power model. We only give the results assuming that power is scaled linearly with port usage, except that unused units dissipate 10% of their maximum power[7].

4.1 Experiment Environment

The target machine includes a five-stage pipelined data path, general registers, I-cache, D-cache, integer ALU, float ALU, control units and so forth. The configuration of the processor is shown in Table 1.

Table 1 Configuration of SimpleScalar.

Issue Width	4	l1 data cache config,	dl1:128:32:4:1
Window Size	16	l2 data cache config,	ul2:1024:64:4:1
Number of Virtual Registers	32	l1 inst cache config,	il1:512:32:1
Number of Physical Registers	16	memory access bus width (in bytes)	8
instruction decode B/W (insts/cycle)	4	number of integer ALU's	4
instruction issue B/W (insts/cycle)	4	number of integer multi/dividers	1
register update unit (RUU) size	16	number of memory system ports	2
load/store queue (LSQ) size	8	number of floating point ALU's	4
		floating point multi/dividers	1

4.2 Feasibility Validation

The semantics of SimpleScalar ISA, a superset of MIPS, is used as testing instruction set. By instruction profiling, we get the basic power and switching power of functional components caused by each instruction, which has been depicted in Figure 1. The components include branch predictors, instruction window, load/store queue, registers, result buses, clock buffers and clock wires, ALU, data and instruction caches.

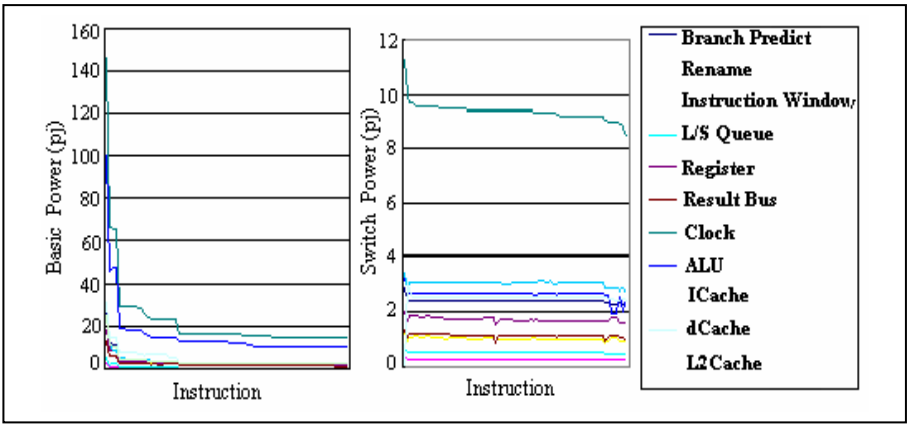


Fig. 1. Basic Power & Switch Power of each functional component

4.3 Accuracy Validation

Benchmarks are used to validate the accuracy of the model. Since for many instructions, the basic power and switching power of their functional components are similar, it is feasible to reckon the usage statistics of each component. We look up the basic power and switching power of the components, and then calculate the total energy consumption using the aforementioned model and the components' usage statistics. Compared with the energy directly measured on Wattch, our model has shown an error of 5% on average and no more than 9% at most, as Table 2 has shown.

Table 2 Test results measured on Wattch and computed by our model.

Benchmark	Power consumption measured on Wattch	Power consumption computed with our model	Error（%）
bubble	412243.15	390737.67	5.23
fibo	17178445	15687847.76	8.68
qsort	9991871.33	9586222.36	4.06
sort	9585787.66	9243115.58	3.57
hello	144890.46	138672.16	4.29
dhry	5958441.76	5572296.74	6.48
sieve	19867955.72	18191993.95	8.44
whetd	21919019.6	21235230.4	3.12
wheds	21899118.5	21362921.6	2.45

5 Advantages and Limits

5.1 Advantages

A limitation of traditional instruction-level power estimation model is the complexity to compute inter-instruction power. For an ISA that has n instructions, there are C^2_{n+1} possible instruction-pairs, which mean the computing complexity is $O(n^2)$ [2]. Our functionality-based model is much simpler. By proposing the idea that the switching power of two components is the sum of the cease stage power of one component and the startup stage power of the other, we reduce the work load to $O(n)$.

The model proposed in this article is accurate and its parameters are easy to get by means of instruction profiling. Although the experiments are made on SimpleScalar/Wattch, a close derivative of the MIPS architecture, the model and the analysis process can easily be applied to other target processors.

5.2 Limitations

The model assumes an ideal situation where pipelines are fully occupied. Practically, instruction correlation is unavoidable and pipeline stalls are expected. Thus power consumption of clock systems must be underestimated since actual executing time is longer. This is an explanation of why the total power computed by our model is on an average 5% less than the power measured on Wattch.

Our model is based on the assumption that the overhead cost for functional components of an instruction is not strongly dependent on the neighboring instructions, but dose depend on whether the neighboring instructions is the same or different. The energy for a particular component is either $B(i, j)$ or $B(i, j) + S(i, j)$ depending on the previous instruction. Actually, switching power of components is dependent on the previous instruction. Yet considering that grouping instructions into clusters also decreases accuracy, this should compare favorably [3].

Some components may startup and stop during the execution of instructions and their switching power is ignored. Fortunately, the phenomenon seldom happens when

one instruction is repeatedly executing. Even if it does happen, the switching power would be compensated in the process of computing basic power.

6 Conclusions and Future Work

This paper describes a functionality-based instruction-level power analysis technique. By considering the program executing process as hardware components changing between working state and sleeping state, we use NOP instructions to model transitions between the two states. Experiments show that our model is able to estimate the energy consumption within 95% accuracy on the average.

The model reduces workload of computing switching power between instruction pairs. It is also convenient to have a quick glance at the run-time software power from assembly code description.

Future work attempts to take pipeline stalls into account and implement the model into a tool in our development of THUMP (TsingHua University's Micro Processor) and its operation system TUESLinux (Tsinghua University Embedded System Linux), whose ISA is an derivative of MIPS 4KC. More data are to come and we would evaluate our model on the developing board.

References

1. Tiwari Vivek, Malik Sharad, and Wolfe Andrew. "Power analysis of embedded software: A First step towards software power minimization." IEEE Transactions on VLSI Systems, vol. 2, no. 4, Dec. 1994, pp. 437-445
2. Mehta Huzefa, Owens Robert Michael, and Irwin Mary Jane. "Module energy characterization using clustering." Proceedings of Design Automation Conference, June 1996.
3. B.Klass, D.Thomas, H.Schmit, and D.Nagle. "Modeling inter-instruction energy effects in a digital signal processor." In Power-Driven Microarchitecture Workshop, June 1998.
4. D.Burger and T.M.Austin, "SimpleScalar Tool Set", Univ. of Wisconsin-Madison Computer Science Dept., Tech, Report #1342, June, 1997.
5. M. Lajolo, A.Raghunathan, S.Dey and L.Lavagno, "Efficient power co-estimation techniques for system-on-chip design" In Proc. Design and Test Europe, Mar 2000, pp. 27-34
6. G. Qu, N. Kawabe, K. Usami, and M. Pothonjak, "Function-level power estimation methodology for microprocessors" In Proc. Design Automation Conf. June 2000, pp. 810-813
7. D.Brooks, V.Tiwari, and M.Martonosi "Wattch: A Framework for Architectural Level Power Analysis and Optimizations" Princeton Univ.
8. Anantha P. Chandrakasan, Amit Sinha, "JouleTrack: A Web Based Tool for Software Energy Profiling", Proc. 38th Conference on Design Automation, June 2001, pp. 220-225
9. H. Mehta, R. M. Owens, M. J. Irwin, "Instruction Level Power Profiling", ICASSP'96, pp. 3326-3329

Robust and Adaptive Dynamic Power Management for Time Varying System[1]

Min Li[1,2], Xiaobo Wu[1], Menglian Zhao[1], Ping Li[2] and Xiaolang Yan[1,2]

[1] Institute of VLSI Design, Zhejiang University
[2] School of Information Science and Engineering, Zhejiang University
limin@imec.be

Abstract. Dynamic Power Management (DPM) is an effective power reduction technique to dynamically control power state of system components. Although there are already a lot of papers on DPM, few of them present robust and adaptive solution to handle the inherent time varying behavior of real world system. In this paper, we propose techniques for targeting DPM on the time varying behaviors. In our approach, we apply efficient policy optimization method to generate online power management policy to achieve adaptiveness. Moreover, in order to handle the small scale varying behavior (perturbation) that tends to significantly degrade the performance of DPM, we apply the Bounded Parameter Markov Decision Process and Interval Value Iteration to derive robust policy tolerant to perturbations. Simulation results show that the proposed techniques are effective.

1 Introduction

Rapid growth in the demand for portable, battery-operated electronics for communications, computing and consumer applications, as well as the continued scaling of VLSI technology, has begun to alter significantly the power constraints under which the systems are designed. The battery life, weight, and volume needed to supply energy to a portable electronic system are typical dominant considerations in the design of that system. However, improvements in battery capacity have not kept pace with the development in microelectronics technology. Consequently, efficient energy utilization becomes one of the key challenges faced by the system designer.

As well known, Dynamic Power Management (DPM) can achieve a great power reduction by controlling performance and power levels of electronic systems in which, a Power Manager (PM) monitors the overall system states, and controls the power state of system components. The general framework for DPM is depicted in Fig. 1 [1] [2]. The Service Requester (SR) generates requests for service, which are queued in buffer first, and then handled by Service Provider (SP); the PM observes states of SP, SR as well as buffer, and then decides which power mode of SP to use.

[1] This work is supported by the National Natural Science Foundation of China under grant No. 90207001.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 449-455, 2005.

The unavoidable problem in DPM is that power state changing (e.g., spin up and down a disk drive, change service rate of a network interface) will impose penalty in terms of power, time and performance. Hence, the balance between potential penalty and power reduction should be judiciously exploited. In the coming deep-submicron era, leakage current becomes the dominant source of power consumption in semiconductor devices, while DPM is the most effective way to combat with it on the system level.

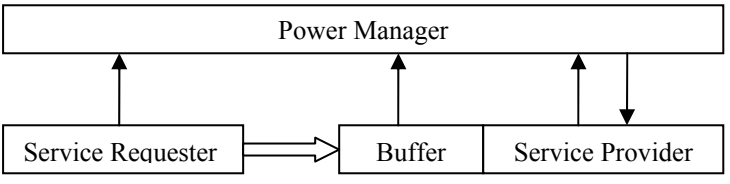


Fig. 1. The power manage system consists of Power Manager, Service Requester, Buffer and Service Provider

A lot of existing DPM algorithms are based on Markov Decision Process (MDP), and most of them assume DPM controlls a completely known stationary stochastic system [1][2][3], i.e., the probability characteristics of Markov chains are completely known in prior and never change over time axis. However, this is not the true case since real world system is usually uncertain and time-varying. Recently, some papers have studied how to handle the uncertainty and nonstationary characteristics of the real world, among which, [4] one proposes a method for nonstationary service requests. In this approach, power management policies are optimized offline on a set of possible parameter vectors using linear programming; and, when running online, the transitioning probability of system components are estimated by Maximum Likelihood Estimation (MLE), and then the most similar parameter vectors will be looked up from pre-encoded table, based on which the power management policy, generated by linear interpolation,[5] extends the work by designing a more delicate mode-switching controller. However, those methods are effective only for a small-scale system. For a moderate scale system, say, 500 states and 5 actions, it is infeasible to partition parameters into fine-grained intervals and perform policy optimization in prior. In addition, the policy generated from linear interpolation is imprecise. Furthermore, with a mode switcher, the existing methods can track only large scale varying behavior. Actually, small variations called perturbations in this paper are also very likely to happen, and parameters estimation itself is inherently imprecise. Some existing work has shown that the performance of MDP is very sensitive to such kind of perturbations [6][7]. Hence, the perturbation also needs to be judiciously handled.

In this paper, firstly, we present the experimental results of implementing a number of policy optimization methods, namely *linear programming*, *policy iteration* and *value iteration*. It is shown that the widely discussed linear programming runs extremely slow in practice, and is not suitable for online policy optimization. On the contrary, policy iteration and value iteration, which are designed specifically for MDP, consume a great deal less time, hence are suitable for online policy optimization by which, the large scale varying behavior can be tracked. Secondly, we apply

Bounded Parameter MDP (BPMDP), which is a generalization of MDP and assigns a real interval instead of an exact figure to the transition probability of Markov chain, to combat the small scale varying behaviors and achieve robustness. Finally, simulation results are shown in section 4, and concluded in section 5 that our method is indeed effective for time varying systems.

2 DPM and Online Policy Optimization

2.1 MDP and Formulation for DPM

The key concept in MDP is Stationary Controllable Markov Chain (SCMC) $M(a)$, which is actually a Markov chain with state set Φ and transition probability controlled by command a. Denote the command set by $A = \{a_i\}$, element of the transition probability matrix P by $p_{s_i,s_j}(a) : A \to [0,1]$, which does not vary with time.

The controller of SCMC is the procedure that issues command $a \in A$ at the beginning of each time interval according to history of the system. If the decision depends only on current system state, it is called Markov decision. A policy $\pi \equiv [\delta_1, \delta_2, ...)$ is a sequence of decisions at discrete intervals; π is called stationary policy if $\delta_i = \delta_j$ for $\forall i$ and $\forall j$. If the policy is Markovian stationary policy, each decision can be represented by a vector $\delta = (\delta_{\phi_1}, \delta_{\phi_1}, ...\delta_{\phi_{|\Phi|}})$, where δ_ϕ is a vector of probability for issuing action $a \in A$ when system is in state ϕ. In a more formal way

$$\delta_\phi = \{\Pr(a \mid \phi), s.t. a \in A\},\ \Pr(a \mid \phi) : \Phi \to [0,1],\ \sum_{a \in A} \Pr(a \mid \phi) = 1. \quad (1)$$

Deterministic Markovian policy is a special case of above model, in which $\Pr(a \mid \phi) : \Phi \to \{0,1\}$. We consider stationary deterministic Markovian policy in this paper.

The MDP is a composition of a SCMC, and a reward function: $r : \Phi \times A \to R$. Formally, MDP is a tuple $M \equiv (\Phi, A, P, r)$. The reward function reflects interests for specific MDP. In the context of DPM, the reward is power reduction or a function of power reduction.

The expected value function, denoted by $V_\pi(\phi)$, is associated with each state of MDP. The $V_\pi(\phi)$ maps each state to its expected discounted cumulative reward defined by

$$V_\pi(\phi) = r(\phi, \pi(\phi)) + \beta \sum_{\phi' \in \Phi} V_\pi(\phi') p_{\phi',\phi}(\pi(\phi)), \quad (2)$$

where $0 \le \beta \le 1$ is the discount factor. The optimal value function V^* is defined as

$$V^*(\phi) = \max_{a \in A} (r(\phi, a) + \beta \sum_{\phi' \in \Phi} V_\pi(\phi') p_{\phi',\phi}(a)) \quad (3)$$

An optimal policy is any policy π^* for which $V^* = V_{\pi^*}$. Every MDP has at least one optimal policy.

Generally speaking, when solving DPM based on MDP, the SCMC model of the whole system is a composition of a set of Markov chains and SCMCs. Specifically, the system level of SCMC consists of SP, SR and buffer, among which SP is a Markov chain independent of any other components; SR is a SCMC with control set $A = \{a_i\}$; the buffer is a SCMC controlled by SR and SP. Hence, the state space of the complete system is $\Phi \equiv S \times D \times Q$, where S is the state space of SP, D is the state space of SP, and Q is the state space of buffer. The details of MDP and DPM formulation are omitted here due to space limitation. Related information can be found in [1][2].

2.2 Efficiency of Online Policy Optimization

The stationary policy δ_ϕ can be optimized via a number of ways, e.g., Linear Programming (LP), Policy Iteration (PI) and Value Iteration (VI) and various variances. LP is adopted in most previous papers because the QoS (Quality of Service) constrained energy optimal policy optimization can be easily formulated as a LP problem. Unfortunately, since the state space of DPM system usually is large, LP runs extremely slow in most cases. Hence, in [4][5], policies are optimized offline. As discussed above, it is very hard to take into account all possible parameter configurations when optimizing policy offline. Hence, we prefer online policy optimization that can indeed deal with various situations in a real system.

In order to evaluate the efficiency, i.e., the required computation time, of online policy optimization, we implement LP, PI, and VI with ANSI C and build executable file targeting on x86 personal computer, then measure the time needed by policy optimization with different problem scales. We have found that PI and VI consume only less than one second for a moderate scale MDP (500 states, 5 actions), and it is feasible to implement online policy optimization. There are many methods to decide when to perform online policy optimization, and the simplest and effective way is to perform it periodically.

However, not like LP, PI and VI can deal with only one objective function. Hence, we use a linear projection of all metrics (power consumption, delay, loss rate, etc.) as the objective function. Actually, the weighted sum of metrics is similar to the proportional QoS, which is very practical and becomes very popular in recent years [8].

3 Perturbation -Tolerant Policy Optimization

As discussed above, the real world is inherently time varying. It's impossible to account for all variations in the periodical online policy optimization, because the optimization period has to be long enough to estimate parameter (transition probability) accurately. Between two consecutive policy optimizations, perturbations, i.e., small

scale variations, are very likely to happen. Hence, during policy optimization, we need to derive a policy that is tolerant of these possible perturbations.

We adopt the BPMDP for perturbation tolerant online policy optimization. The BPMDP is proposed in [9], and the key idea is to assign a closed interval to transition probability and reward function, so that uncertainty can be taken into account. Formally speaking, BPMDP is a tuplet $M_B \equiv (\Phi, A, \hat{P}, \hat{r})$. $\hat{P}(a)$ and $\hat{r}(\phi, a)$ are closed real interval, in another word, $\hat{P}(a) = [P_{\min}(a), P_{\max}(a)]$, $\hat{r}(\phi, a) = [\hat{r}_{\min}(\phi, a), \hat{r}_{\max}(\phi, a)]$. Consequently, $\hat{V}$ is also a closed real interval,

$$\hat{V}_{\pi}(\phi) = [\min_{M \in M_B} V_{M,\pi}(\phi), \max_{M \in M_B} V_{M,\pi}(\phi)] \quad . \tag{4}$$

More important, [9] proposed a slightly modified variant of value iteration algorithm. It is proved that, for fixed discount rate β, the iteration algorithm converge to the optimal value function in a number of steps is a polynomial express of the number of states, actions, and bits used to represent the MDP parameters. Due to the space limitation, details of the algorithm and proof are omitted herein; those who need them please refer to the journal paper [10].

4 Simulation Result

The system to be simulated is a mobile device that has certain data to deliver to a base-station via wireless communication channel. According to the general model depicted in Fig.1, SR is a Markov modulated traffic source that can represent various data traffic, like H.263, MPEG4, Telnet, FTP, etc.[11]; SP is a wireless transmitter that has a number of power states. The wireless channel quality herein is assumed to be constant. In real world system, mobile device users often launch different applications alternatively, and this is simulated by adjusting dynamically related SR parameter. Moreover, we apply small scale white noise to the parameters to model small scale varying.

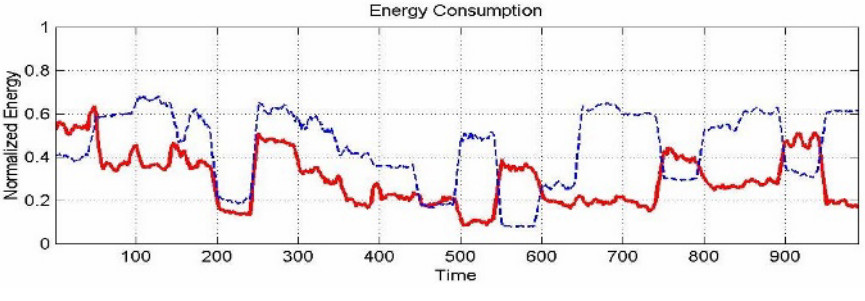


Fig. 2. Comparison of normalized energy consumption

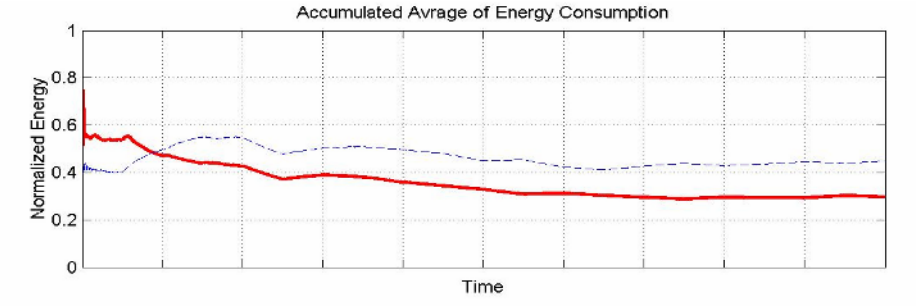


Fig. 3. Comparison of accumulated energy consumption

The plot of reduced energy consumption shown in Fig. 2 is normalized to the energy consumption without DPM. In this figure, the dashed line is the result of offline optimization and online policy interpolation; the solid line is the result of our method. Apparently, the proposed method significantly outperforms the existing one in most cases. Moreover, we plot the accumulated average energy consumption in Fig. 3 to study the long term behavior. It is shown that the proposed method consumes about 30% energy, while the existing approach consumes about 45%. Hence, 50% improvement is achieved.

5 Conclusion

As an effective technique of power reduction, DPM is becoming more and more important. In this paper, we have proposed a new DPM techniques targeting on the time varying and uncertain behavior of real world embedded systems which has not been extensively studied yet. The proposed techniques are adaptive and robust. Specifically, the adaptiveness is achieved by efficient online policy optimization, and the policy is derived from BPMDP so that it is tolerant of perturbations. Simulation results show that the proposed techniques can indeed outperform existing one.

References

1. Qiu, Q., Qu, Q., Pedram, M.: Stochastic Modeling of a Power-Managed System-Construction and Optimization. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Vol. 20. Oct. (2001) 1200–1217
2. Benini, L., Bogliolo, A., Paleologo, G.A., De Micheli, G.: Policy Optimization for Dynamic Power Management. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Vol. 18. June (1999) 813–833
3. Benini, L., Bogliolo, A., De Micheli, G.: A Survey of Design Techniques for System-Level Dynamic Power Management. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Vol. 8. June (2000) 299–316

4. Eui-Young Chung, Benini, L., Bogliolo, A., Yung-Hsiang Lu, De Micheli, G.: Dynamic Power Management for Nonstationary Service Requests. Computers, IEEE Transactions on, Vol. 51. Nov. (2002) 1345–1361
5. Ren, Z., Krogh, B., Marculescu,,R.: Hierarchical Adaptive Dynamic Power Management, in Proc. Design, Automation and Test in Europe Conf., Paris, France, Feb. (2004)
6. Altman, E., Schwartz, A.: Adaptive Control of Constrained Markov Chains Automatic Control. IEEE Transactions on, Vol. 36. April (1991) 454–462
7. Ren, Z., Krogh, B.H.: Adaptive Control of Markov Chains with Average Cost. Automatic Control, IEEE Transactions on, Vol. 46. April (2001) 613–617
8. Yang Chen, Chunming Qiao, Hamdi, M., Tsang, D.H.K.: Proportional Differentiation: a Scalable QoS Approach. Communications Magazine, IEEE, Vol. 41. June (2003) 52–58
9. Givan, R., Leach, S., Dean, T.: 1997. Bounded Parameter Markov Decision Processes. In Steel, S., and Alami, R., eds., Proceedings of the 4th European Conference on Planning (ECP-97): Recent Advances in AI Planning, Berlin: Springer Vol. 1348 of LNAI, 234–246
10. Robert Givan, Sonia Leach, Tom Dean: Bounded-parameter Markov Decision Processes. Artificial Intelligence, Vol. 122. (2000) 71-109
11. Daniel P. Heyman, David Lucantoni: Modeling Multiple IP Traffic Streams with Rate Limits. IEEE/ACM Transactions on Networking (TON), December (2003)

Skyeye: An Instruction Simulator with Energy Awareness[1]

Shuo Kang, Huayong Wang, Yu Chen, Xiaoge Wang, and Yiqi Dai

Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, P.R.China
{ks02, wanghy02}@mails.tsinghua.edu.cn,
{yuchen, wangxg}@mail.tsinghua.edu.cn,
dyq@theory.cs.tsinghua.edu.cn

Abstract. This paper presents a novel strategy aimed at modeling the instruction energy consumption of ARM microprocessors with dynamic voltage scaling (DVS) support. A novel energy estimation algorithm is designed, which can record the function calls, and generate a detailed energy profile for each function in a specific program. Some of the optimization policies for implementation are also discussed. These optimization policies reduce the workload of the energy estimators for the individual SOC systems. The prototype system, SKYEYE, can automatically detect the voltage/frequency variation activated by DVS system, and adjust the energy estimation model accordingly. The experiment results further prove the effectiveness of the algorithm.

1 Introduction

With the advent of portable and hand-held computing/communication systems, energy consumption has recently become a critical issue in system design. Energy-awared simulator is a key tool to develop energy efficient programs. The development of a simulation environment that can provide useful information to the programmers faces a number of challenges: The first step that must be taken is to develop an abstract energy model. The model must provide a simplified but accurate picture of how energy consumption is related to the actions of an application. In this way, any causality between the application source code and the energy consumption of the device can be exploited. The next step is to design experiments that accurately characterize the energy consumption of the device in terms of the model. These values are then used as parameters in a simulator incorporating the abstract energy model.

There is so much research work that tries to address the problem of energy saving from the hardware level, such as low-power circuits or processors [1] and voltage scaling techniques [2]. Though it is true that maximum energy savings are possible through hardware optimizations, this paper tries to consider this problem from the

[1] This research was supported by National 863 project of China (No. 2003AA1Z2090) and National Science Foundation of China (No. 60203024).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 456-461, 2005.

software aspect. This paper proposes a novel energy estimation algorithm and optimization policies for its implementation. According to this algorithm, a prototype system, named SKYEYE, is downloadable from website http://www.skyeye.org, which incorporates energy models for many different kinds of ARM architectures and various peripheral devices, such as NIC and LCD.

2 Related Researches

Vivek Tiwari develops an instruction level energy model for Intel 486DX2 and Fujitsu SPARClite 934 in paper [3]. The basic idea is the sum of the energy costs of each instruction that is executed in a program can be estimated for the energy cost of the whole program. Jeffry T. Russell believes that there is actually no need to consider the individual assembly instructions to accurately predict the energy consumption [4]. The energy consumption can be predicted by using the processor average power consumption multiplied by the software execution time. This easier method accurately predicts energy consumption with 99% confidence based on physical measurements. Paper [5] presents an instruction of class profiling technique, which has an estimated error of less than 3% with trivial runtime overhead. Figure 1 is the data provided by paper [5].

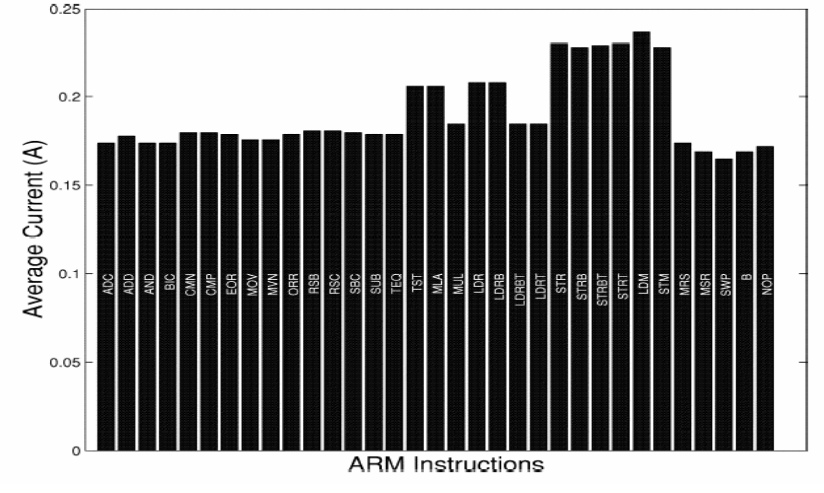


Fig. 1. Current of ARM instructions

With the improvements of the research on energy model, quite a lot of simulators appear in the field of embedded systems. CycleSim [6] is an event and execution-driven simulation engine for the PowerPC architecture. It is developed to provide fast, cycle-accurate simulation of operating systems and applications. Paper [7] describes a suite of simulation tools for the PalmOS family. And paper [8] illustrates a way to estimate SOC.

Most of the traditional work mentioned above focuses on the processor unit, not the overall system. The effect of a model on the overall system energy consumption is more important than its effect on the particular component it concerns. And, the support to DVS system has never been taken into consideration. That is, it is always assumed that the voltage or frequency is constant during runtime. However, more and more embedded systems have the ability to change the processor's voltage or frequency to save energy. The major source of energy consumption in digital CMOS circuits is the dynamic power dissipation, which is computed by formula:

$$P = C_a \times N_{sw} \times V_{dd}^2 \times f \quad , \tag{1}$$

where C_a is the output capacitance, N_{sw} is the number of switches per clock, V_{dd} is supply voltage, and f is the processor clock frequency. Processor clock frequency is almost linearly related to V_{dd} as following:

$$f \approx k \times \frac{(V_{dd} - V_{th})^2}{V_{dd}} \quad , \tag{2}$$

where k is a constant, and V_{th} is the threshold voltage. In practice, the voltage V_{dd} will decrease by slowing down the clock. And the energy consumption, which is proportional to the square of the voltage, will usually be reduced. By these means, DVS system is widely researched and becomes a challenge for the designers of energy-awared simulators.

3 Energy Model

In this paper, it is assumed that voltage scaling has a fixed discrete value domain, defined as $S=\{V_1,...,V_m\}$. The instruction set of the ARM architecture is denoted as $ISA=\{A_1,..., A_n\}$.

Definition 1: Instruction Average Current Matrix (IACM) is a matrix to record the average current of each instruction at each supply voltage.

$$IACM = \begin{pmatrix} I_{11} & I_{12} & \cdots & I_{1n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ I_{m1} & I_{m2} & \cdots & I_{mn} \end{pmatrix}, \tag{3}$$

where I_{ij} denotes the average current of instruction A_j at voltage V_i.

Definition 2: Instruction Execution Time Vector (IETV) is a vector to record the total execution time of each instruction A_j in a program segment.

$$IETV = (N_1 \times M_1 \times \tau, N_2 \times M_2 \times \tau, \cdots, N_n \times M_n \times \tau)^T , \tag{4}$$

where M_j denotes how many instruction A_js are executed in the program segment, N is the number of execution cycles for one A_j, and τ is the clock period.

Definition 3: Voltage Selection Vector (VSV). Since different segments may adopt different voltages, VSV describes which voltage is selected by current segment.

$$VSV = \{0, \cdots, V_k, \cdots, 0\} \quad . \tag{5}$$

VSV has *m* elements, but only one can be nonzero. If the *k*th element is nonzero, the *k*th element must be equal to V_k, which means the current segment selects supply voltage V_k.

The energy consumption of one program segment can be estimated by the segment's *IETV* and *VSV*.

$$E_{seg} = VSV_{seg} \times (IACM \times IETV_{seg}) \quad . \tag{6}$$

And the total energy of the whole program in a DVS system is estimated by:

$$E = \sum_{seg} E_{seg} \quad . \tag{7}$$

In formula (6), *IETV* and *VSV* are online recorded by simulator during runtime; *IACM* is provided by offline measuring the instruction current on a specific target hardware platform.

In our energy model, we have considered another problem. Even the same instruction may have different energy consumption with different arguments. For example, LDR/STR instruction can access I/O address space, which usually involves various peripheral devices to work and the energy consumption varies greatly. In SKYEYE, several peripheral devices are simulated and multiple device states are defined for each of them. Different device states have different energy models. The transition cost between states is computed by a fixed value of energy cost, which is obtained by experiments on the target hardware.

4 Optimization Policies

The optimization policies include "macro-modeling" and "energy and delay caching". The idea of macro-modeling is derived from RTL hardware energy estimation [9]. Software macro-modeling refers to the pre-characterization of a comprehensive set of high-level macro-operations in terms of various metrics such as code size, performance and energy. For example, a macro-operation could be an arithmetic operation with assignment to a variable, an emission of an event, and etc. Characterization process is performed by compiling each macro-operation down to a sequence of assembly-level instructions for the target processor, and computing its energy dissipation using an instruction-level simulator. If the energy consumption of a macro-operation is known as a priori, the energy estimation can be based on macro-operation, rather than single instructions.

Energy and delay caching technique comes from our observations to runtime systems. In our experiments of energy estimation, we have observed that a few "paths of computation" in the software components are executed for a large number of times. This conforms to the empirical observation that a small fraction of the code accounts

for a large part of the total execution time. In general, it is possible that each execution of a block of code results in a distinct delay and energy consumption. However, in practice, we have also observed that the number of distinct energy and delay values for a single code block is much smaller than expected. SKYEYE exploits the above observation to significantly enhance the efficiency of energy estimation.

5 Experiments and Results

We run five applications to compare their energy costs in a DVS environment. The following table records the parameters for each application.

Table 1. Five applications in DVS system

	APP1	APP2	APP3	APP4	APP5
Instructions	25175079	25053189	25095178	23904860	29100672
Cycles	55148607	54821274	67557386	54048354	72031032
Energy(J)	0.097121	0.096557	0.095776	0.095453	0.110202

The Linux kernel is modified to support DVS. Then, the CPU frequency is scalable during runtime. Table 1 displays the instruction number, cycles and energy recorded by SKYEYE. And Figure 2 compares the energy estimated by SKYEYE and the energy measured on the real hardware platform (sitsang board).

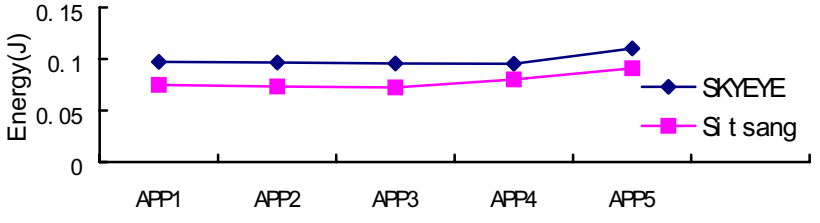


Fig. 2. Compare SKYEYE to hardware

These five applications are also measured in an environment without DVS. Their parameters are recorded in the table 2.

Table 2. Five applications in non-DVS system

	APP1	APP2	APP3	APP4	APP5
Instructions	25175079	25053189	25095178	23904860	29100672
Cycles	55148607	54821274	67557386	54048354	72031032
Energy(J)	0.107347	0.108842	0.105753	0.107843	0.121132

The comparison between SKYEYE and the hardware measurement is recorded in figure 3.

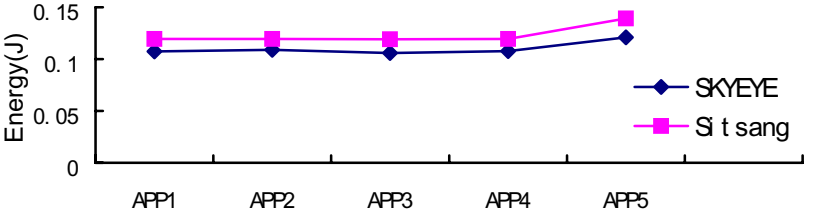


Fig. 3. Compare SKYEYE to hardware without DVS

6 Conclusions

This paper presents a novel energy estimation algorithm for ARM simulators and discusses the optimization policies for its implementation. This new algorithm supports DVS system and considers the problem of peripheral devices. The future work will focus on how to improve the precision of the energy estimation.

References

1. Anantha Chandrakasan, Robert W. Brodersen: Low-power CMOS design. IEEE Press, Piscataway, New Jersey (1998)
2. Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter Glynn, Giovannl De Michell: Dynamic voltage scaling and power management for portable systems. In: Proceedings of Design Automation Conference. IEEE Computer Society Press, California (2001) 524-529
3. Vivek Tiwari, Sharad Malik, Andrew Wolfe: Power analysis of embedded software: a first step towards software power minimization. IEEE Transaction on VLSI Systems. 2(4) (1994) 437-445
4. Jeffry T. Russell, Margarida F. Jacme: Software power estimation and optimization for high performance, 32-bit embedded processors. In: Proceedings of International Conference on Computer Design. IEEE Computer Society Press, California (1998) 328-333
5. Amit Sinha, Nathan Ickes, Anantha Chandrakasan: Instruction level and operation system profiling for energy exposed software. IEEE Transaction on VLSI Systems. 11(6) (2003) 1044-1057
6. Hazim Shafi, Patrick Bohrer, James Phelan, Cosmin Rusu: Event-based system power simulation. In: Proceedings of the IBM Austin Conference on Energy Efficient Design. IBM Press, Austin, Texas (2002)
7. Cignetti Todd L., Komarov Kirill, Ellis Carla Schlatter: Energy estimation tools for the Palm. In: Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems. ACM Press, Boston Massachusetts (2000) 96-103
8. Marcello Lajolo, Anand Raghunathan, Sujit Dey, Luciano Lavagno: Efficient power co-estimation techniques for system-on-chip design. In: Proceedings of Design, Automation and Test in Europe Conference and Exhibition. ACM Press, Paris, France (2000) 27-34
9. Tan T. K., Raghunathan A., Lakshminarayana G., Jha N. K.: High-level software energy macro-modeling. In: Proceedings of Design Automation Conference. IEEE Computer Society Press, California (2001) 605-610

The Modeling for Dynamic Power Management of Embedded Systems

Jiangwei Huang, Tianzhou Chen, Minjiao Ye, Yi Lian

College of Computer Science, Zhejiang University,
Hangzhou, Zhejiang, 310027, P. R. China
{hjw, tzchen, yeminjiao, yl}@zju.edu.cn

Abstract. In this paper we present a new modeling technique using software engineering tool Flow Model for modeling and solving the Dynamic Power Management (DPM) with complex behavioral characteristics. Using this tool we can model the whole system easily. Experimental results show that the proposed technique can achieve more than 12% power saving compared to other DPM techniques.

1 Introduction

As mobile computing is getting popular, there is an increasing interest in techniques that can minimize energy consumption. The goal of low-power design for battery-powered devices is thus to extend the battery service life while meeting performance requirements. Incorporating a dynamic power management scheme in the design of an already-complex system is a difficult process that may require many design iterations and careful debugging and validation. [1]

As the processor may not be fully utilized all the time, the variation in system load can be exploited to reduce power dissipation. The processor can be turned off or made to operate at lower speed when the processor has no or little work to do. Some processors allow the voltage to be dynamic adjusted. This is called Dynamic Voltage Scaling (DVS). The relationship among the power consumption rate (P), supply voltage (V_s) and clock frequency (f) can be described by the following formula: $P=C \times f \times V_s^2$, where C is the switched capacitance [8].

We have targeted a more complex power-managed system which is shown in Fig 1. This model shows a typical multi-server and multi-requester system. The system contains multiple SPs (Service provider) with their own Local Service Queue (LSQ). There are multiple SRs (Service Request) that generate the requests that need to be serviced. The Request Queue (RQ) buffers the requests that generate by SRs. The Request Dispatcher (RD) makes decisions about which SP should serve which request. Different SPs may have different power/performance parameters. In real applications, the RD and LSQs can be part of the operating system, while SPs can be multiple processors in a multi-processor computing system or number of networked computers in a distributed computing system. [3][5]

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 462-467, 2005.

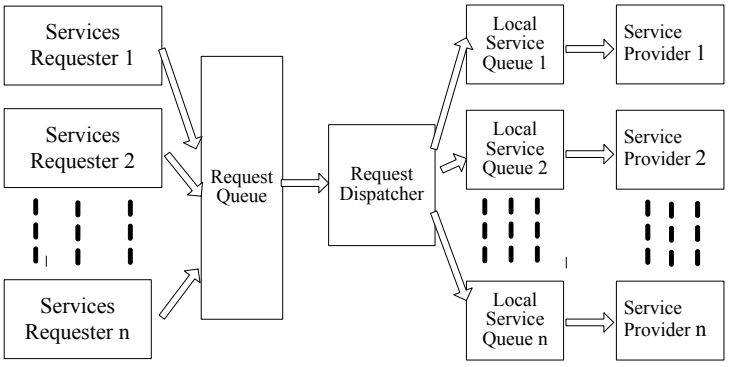


Fig. 1. Multi-server and multi-requester system.

Because of the complex system behaviors that are present. It needs an accurate modeling method to build the whole system. For example, we need to consider the synchronization of LSQs and SPs, the synchronization of the SRs and RQ, the synchronization of the RQ and LSQs, the dispatch behavior of the RD, and so on. In this situation we need new DPM modeling techniques for large systems with complex behaviors. In this work, we first present a methodology based on Flow Model to model complex systems such as the one shown in Figure 1. [2]

2 Modeling System with Flow Model

The whole DPM model can be broken into three major parts as shown in Fig 2. The first part is called Request Producer (RP) which is used for producing request including the SRs and RQ. The second part of the process includes the RD which is used for dispatching the request to the request to a suitable SP. We name this part Request Transfer (RT). The RT is used to send the Request to the suitable SP and adjust the states of SP. The last part is Request Consumer (RC), including the SPs and LSQs, used for offering services. Because the buffer between RP and RT is limited, we can find the relationship of RP and RT is producer and consumer, such as RT and RC. If the buffer between the RP and RT is full, the RP needs to stop producing the requests till there is a free space in the buffer.

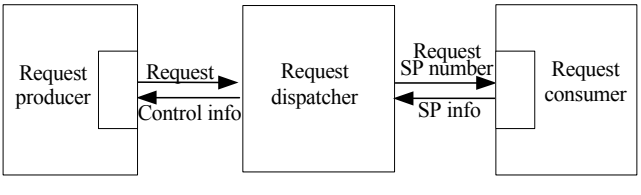


Fig. 2. Three major parts of the whole process

[7] Presents a method to reduce energy consumption by inserting data buffers. The method determines whether power can be reduced by inserting a buffer between two components. This method calculates the length of the period and the required buffer size to achieve the optimal energy savings. With the help of this method, we have designed the size of the buffer between each part. The whole system will be explained by the three single models next.

The single Request Consumer Flow model is given in the Fig 3. We have found that the number of requests generated by the consumer is a Poisson distribution. In the Flow mode, which is graphically represented by a circle in the model is correspondent to a possible server status. A server status is composed by its working mode and its power mode. A working mode can be idle or busy or switching, etc. A power mode can be sleeping or active etc. The server can be in different working modes with the same power mode.

The P_{switch} is correspondent to a random time duration which follows exponential distribution with certain mean value. A Unit Server System (USS) contains a SP and a SQ. The information needed for SP is its power consumption in each power state, its service speed in each power state, and the time and the energy needed to switch from one state to another state. Initially, there is one token in the starting state of this model for the single server. The token switches from one place to another with the status changing of the server. The timing of the token transition from one place to another is decided by the time duration of the activity. [6]

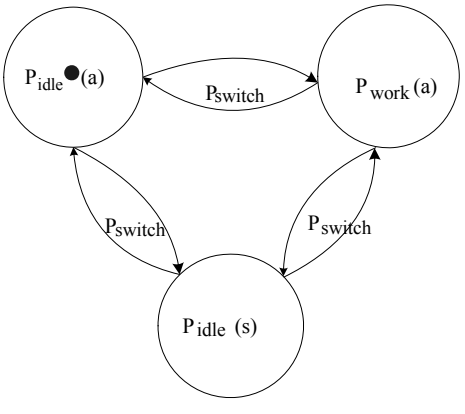


Fig. 3. Flow model of a single Request Consumer

A flow model of a request producer is given in Fig 4. The request generator can generate many types of requests. The time of generating different request is different. We give the definition that the Request type A can only be served by server A, and request type B can only be served by server B, however, request type AB can be served by both server A and B. The Request Generation System (RGS) can generate various types of requests constraint by the probability distribution of request types. Some types of requests can be served by several USS; whereas some other types of requests can be served by only a special USS. If the SQ is full, the RGS will stop generating request. It will resume request generation when there is free space in the SQ.[4]

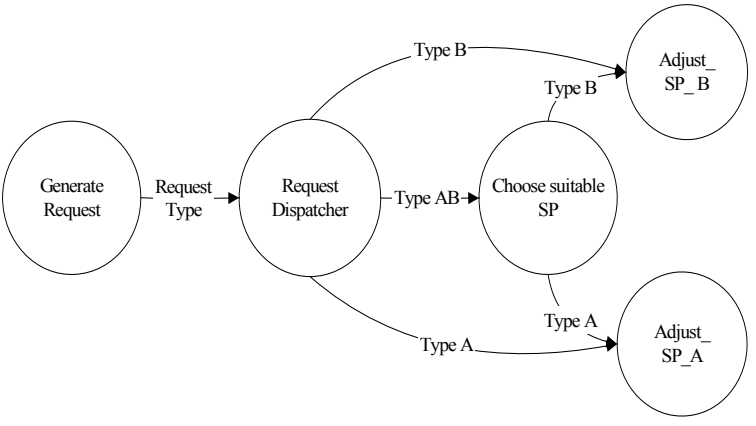


Fig. 4. Request Generation System Modeling

Fig 4 includes the RP and RT. There is one token in Generate Request (GR), it means that the starting state of the system is the request generator. After certain time, the token in GR is switched to Request Dispatcher. After negligible time, the token will be switched to the Adjust_SP_B or Adjust SP A or choose suitable SP.

A multi-server and multi-requester system is a complex system including several USSs and RGSs, many request generators and different interactions among the components. The requests are sending to a suitable server i with certain probability P_i through a dispatcher. If the request can only be served by a server, then $P_i=1$, such as $P_i=0$ means the request cannot be serviced by the server. The probability P_i is controlled by the dispatcher when $P_i \neq 0|1$. It is state dependent and need to be optimized. In the modeling of multi-server system, we must evaluate the function of correlation between the request queue and request generators. If the request queue is full, then the request generator procedure must be stopped. Fig 5 shows the MSS which contains two USSs and one RGS described above.

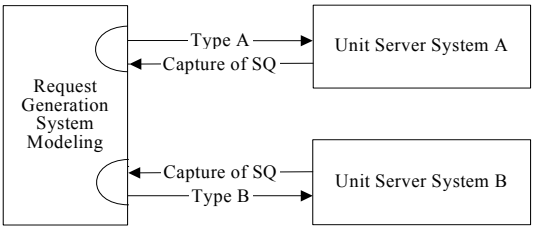


Fig. 5. Multi-server System Modeling

3 Experimental Results

There are two USSs and two RGSs in our MSS. The RGSs are used to generate requests, which can be served by the USS. The requests are queued in the request

queue, and the capacity of the request queue (RQ) is 6. The two USS have the difficult power consumptions and service speeds. The SP in both USSs has three power states: {active, waiting, sleeping}. When the USS was in active state, the USS consume more power than other states. When the USS is in the waiting state, it means there are not many requests being served by this USS. If the USS is in the sleeping state, it means there is no request. The capacity of SQ is 2 [3]. We use the Sistang Board as the USS and RGS. The experimental model is showed in Fig6.

In this setup, we compare the following methods: Optimal USS power management + heuristic dispatch policy, Time-out + heuristic dispatch, Greedy + heuristic dispatch Flow Model-based method.

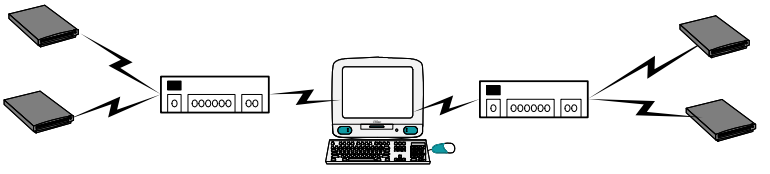


Fig. 6. Experimental Model

Table 1 shows the comparisons of experimental results for the four methods described above.

Table 1. comparisons of experimental results

	vs. Greedy DPM policy	vs.Timeout DPM policy	vs.Local optimal	Average Saved
pA=0.2, pB=0.8	22.75%	13.49%	26.33%	20.856%
pA=0.4, pB=0.6	23.91%	14.99%	23.55%	20.816%
pA=0.5, pB=0.5	24.15%	15.87%	26.63%	22.21%
pA=0.6, pB=0.4	23.01%	14.94%	25.9%	21.28%
pA=0.8, pB=0.2	19.62%	13.65%	44.83%	26.03%

4 Conclusion

From the experimental results, we can find the flow modeling mechanism can save more than 12% power compared to other methods. The improvement shows it is important to build an accurate system model and accurate policy for embedded system. It can save more power than we can expect. We use the Flow model to model and solve the DPM problem for systems with complex behavioral characteristics. It gives an easy way to build the DPM model for embedded system.

References

1. IBM and MontaVista Software: Dynamic Power Management for Embedded Systems, 2003
2. Dexin Li, Qiang Xie and Pai H. Chou Center for Embedded Computer Systems University of California, Irvine: Scalable Modeling and Optimization of Mode Transitions Based on Decoupled Power Management Architecture 2003
3. Qinru Qiu, Qing Wu and Massoud Pedram Department of Electrical Engineering – Systems, University of Southern California: Dynamic Power Management of Complex Systems Using Generalized Stochastic Petri Nets, June 2000
4. M. A. Viredaz and D. A. Wallach: "Power evaluation of a handheld computer", IEEE Micro, vol. 23, no. 1, Jan./Feb. 2003.
5. Q. Qiu, Q. Wu, M. Pedram, "Stochastic Modeling of a Power-Managed System: Construction and Optimization", Proceedings of the International Symposium on Low Power Electronics and Design, pp. 194-199, Aug. 1999.
6. Balakrishnan, N. and Basu, A. P. "The Exponential Distribution: Theory, Methods, and Applications". New York: Gordon and Breach, 1996
7. "Dynamic Power Management Using Data Buffers" Le Cai and Yung-Hsiang Lu Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04) 2004
8. Li, Dexin, Xie, Qiang, and H, Pai. Chou Center for Embedded Computer Systems University of California, Irvine: Scalable Modeling and Optimization of Mode Transitions Based on Decoupled Power Management Architecture 2003

Why Simple Timeout Strategies Work Perfectly in Practice?

Qi Wu and Guang-ze Xiong

College of Computer Science & Engineering
University of Electronic Science & Technology of China
Chengdu 610054, China
{Henrywu,Gzxiong}@uestc.edu.cn

Abstract. In all kinds of *dynamic power management* (DPM) policies, the timeout policy works perfectly in practice, but this could not be elucidated in the traditional queuing theory. With the pareto distribution, the drawback of the exponential distribution was overcome, and this phenomena was explained well. The conclusion provided the theoretical basis for the simplification of DPM algorithm. In this paper, a simple DPM algorithm was described when the length of idle time follows the Pareto distribution. The simulation result testified the conclusion above.

1 Introduction

Battery powered portable appliances impose tight constraints on the power dissipation of their components. Such constraints are becoming tighter as complexity and performance requirements are pushed forward by user demand. Reducing power dissipation is also a design objective for stationary equipment, because excessive power dissipation implies increased cost and noise for complex cooling systems.

One of the most successful techniques employed by designers at the system level is *dynamic power management* (DPM) [1]. This technique reduces power dissipation by selectively turning off system components when they are idle. In practice, the most common power management policy at system level is the *timeout policies* [2,3,4], which often reduce a great deal of the energy consumption of computer devices. But the devices seem to waste energy when waiting for the timeout expire. This inefficiency impels the exploration of more effective techniques. *Predictive policies* shut down a device as soon as an idle period begins when the predicted idle period is long enough to equalize the cost of shutting down and later reactivating the system [5,6,7]. The *stochastic policies* take the DPM problem as a stochastic optimization control problem [8,9,10,11]. It guarantees the optimal results. But its own overhead is very high when the stochastic models are used directly.

What is the optimal policy of DPM? With stochastic models, Wu [12] has proved that the DPM optimal policy is a deterministic Markov policy, namely timeout policy. But if the arriving process of service requests was taken as a Poisson flow and the interval of arriving time obeyed the exponential distributions, the practical effect of the timeout policy could not be explained with traditional queuing theory. Why a simple timeout strategy works perfectly in practice?

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 468–473, 2005.

In recent years, a series of experimental researches have shown that network traffic has obvious *self-similarity* [13,14].Both service and idle time fit heavy-tailed distributions [10]. And this is also true for the length of files [15] and lifetime of processes [16]. These all show the universality of self-similarity of service request. The self-similarity of the sequences implies that their probability distribution function scales in a well-defined way when we change the time scale over which the sequence is calculated. Its autocorrelation function $r(k) = E[(\tau_t - \mu)(\tau_{t+k} - \mu)]/\sigma^2$ has the nature of $\lim_{k\to\infty} r(k) \sim k^{-\beta}$, and the sequence is long-range dependent. Unlike the exponential distribution, the heavy-tailed distribution has the memory characteristics, and its variance and average value are not sure to exist, making it difficult to study self-similar time sequence with mathematics. Some researches on self-similarity were reported in the area of net-traffic [13] and process schedule [17], but there is no report about its effect on the DPM algorithm.

2 System Model

The hardcore of a DMP system is *power manageable components* (PMCs), which has multiple low power states in the idle time. A PMC can be abstracted as a state machine, and transforms its power state in response to the idle events or state switch commands from Power Monitor. Suppose the random variable τ is the length of idle time and its distribution function is $F(t)$. A series of τ make up a random variable sequence $T = (\tau_0, \tau_1, \tau_2, \ldots)$. When τ_i is *independent identically distributed*, [12] has proven that the DPM optimal strategy is a deterministic Markov control strategy.

Suppose PMC has K low power states, which are signed with $z_1, z_2, \ldots, z_k$, from the highest holding power to the lowest; the state of *idle* is z_0 and the power of state z_k is w_k; the mean power and time switching from state z_k to z_l are e_{kl} and λ_{kl}, and from state z_k to active state are e_k and λ_k; the DPM deterministic Markov control algorithm is: if the idle period is larger than κ_k, the state is switched to z_k. Assume $\kappa_0 = 0$ and $\kappa_{k+1} = +\infty$. The expectation of the total power consumption of this algorithm can be expressed as Lebesgue integral:

$$E = \int_0^{\kappa_1} w_0 t dF(t) + \sum_{k=1}^{K} \int_{\kappa_k}^{\kappa_{k+1}} w_k(t - \kappa_k) + e_k + \sum_{l=0}^{k-1} (w_l(\kappa_{l+1} - \kappa_l) + e_{l,l+1}) dF(t) \tag{1}$$

So the DPM problem is transformed into the problem of the optimization of the multi-parameters function:

$$E = \min_{0 \le \kappa_1 \le \kappa_2 \le \ldots \le \kappa_K} f(\kappa_1, \kappa_2, \ldots, \kappa_K) \tag{2}$$

3 DPM for Self-Similar Requests

In recent years, a series of experimental researches showed most of the computer service requests have self-similarity, and the interval of these requests follows heavy-tailed distribution, which has the following characteristic:

$$\frac{\lim_{x\to+\infty} P(X>x)}{x^{\alpha}} = c, 0<\alpha<2 \tag{3}$$

The Pareto distribution is a representative heavy-tailed distribution that is sometimes used to model the distribution of wealth in individuals and value of oil reserves. Its *cumulative distribution function* is $F(x) = 1-(c/x)^{\alpha}$, $c>0$ and $\alpha \in (0,2)$.The *generalized Pareto distribution* arises in one of the key limit theorems in extreme value theory, and it is usually used to model the distribution of excess value over a high threshold in the *Peak Over Threshold* (POT) methodology. Referring to POT methodology, we assume the idle time length fits the Pareto distribution in this paper.

Fig.1 shows the relationship between E and κ_k when the idle time length fits the exponential and Pareto distribution respectively. There is no extremum point in the exponential distribution situation [18], and this can also be proven theoretically. Obviously, the exponential distribution can not expound why a simple timeout strategy can reduce much energy consumption of PMCs. Unlike exponential distribution, in Pareto distribution situation, there is an extremum point. The practical effect of the time-out strategy is well interpreted with Pareto distribution.

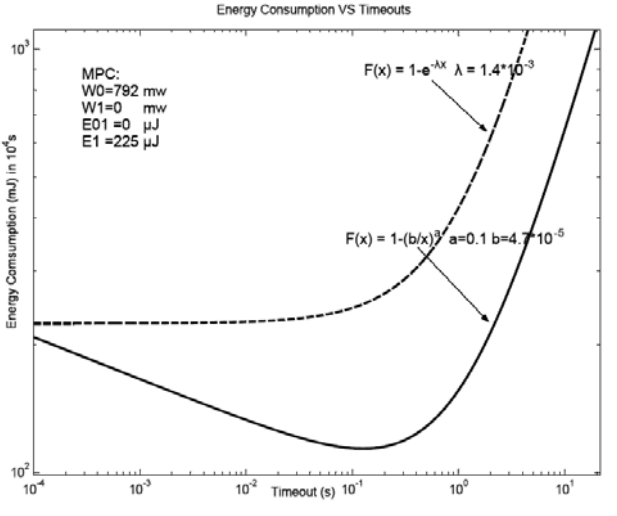


Fig. 1. The relationship between switch time and the expectation of the total energy consumption

Now the DPM problem is transformed to the one of parameter estimation and decision control. The minimum interval of service request might be very short, but the excessively short interval is insignificant to DPM. Similar to the POT methodology, we took $T_{be}/10$ (T_{be} was described in [10]) as the threshold value of the length of idle periods, namely the scale parameter c of Pareto distribution is $T_{be}/10$. The tail index α can be estimated using the *Maximum Likelihood* (ML) method. If the window size is n, the value of α can be calculated:

$$\hat{\alpha} = \left(\sum_{l=1}^{n} \log X_l - \log c \right)^{-1} \tag{4}$$

4 Experimental Results

For the sake of validating the above analytical results, we did a simulation experiment of the DPM algorithm with a hard disk drive (HDD), which has four power states when idling: Idle, LPidle, Standby, and Sleep. Idle state spells the HDD is idle, but keeps all electrical and mechanical parts ready for reading or writing HDD data. The holding power is a little higher and the activation time is almost 0. In LPidle state some HDD electrical parts are shutdown, but disk keeps spinning, so the holding power is less and the activation time is longer than those of the idle state. In Standby and Sleep states, as the disk stop spinning, it takes longer time and more power to activate the disk.

Both e_{ij} and $\lambda_{ij}(i < j)$are 0, and the subscripts 0, 1, 2, and 3 represent Idle, LPidle, Standby, and Sleep, respectively. Other relational parameters are:

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 1.0 \\ 0.8 \\ 0.3 \\ 0.1 \end{pmatrix} Watt, \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.04 \\ 2.2 \\ 6.0 \end{pmatrix} Joule, \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.04 \\ 2.2 \\ 6.0 \end{pmatrix} second \tag{5}$$

For comparison, two distributions, exponential and Pareto, which the length of idle time obeys were studied, and the mean of them are same. The performances of DPM timeout strategy under two distributions were shown in Fig.2 and Fig.3 respectively, when only one low power state of HDD was used. According to the simulation results we may conclude:

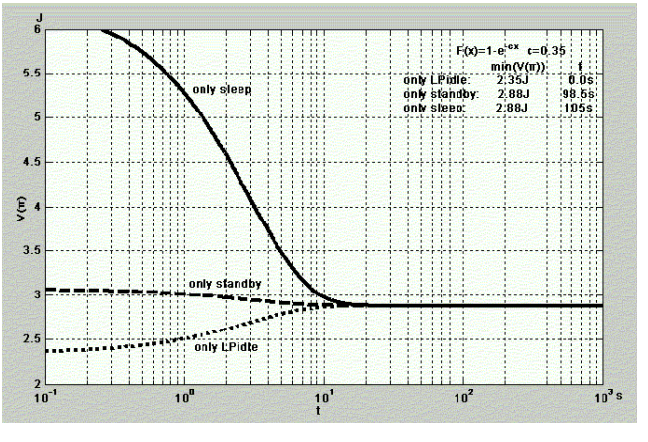


Fig. 2. The relationship between switch time and the total energy consumption in the exponential distribution

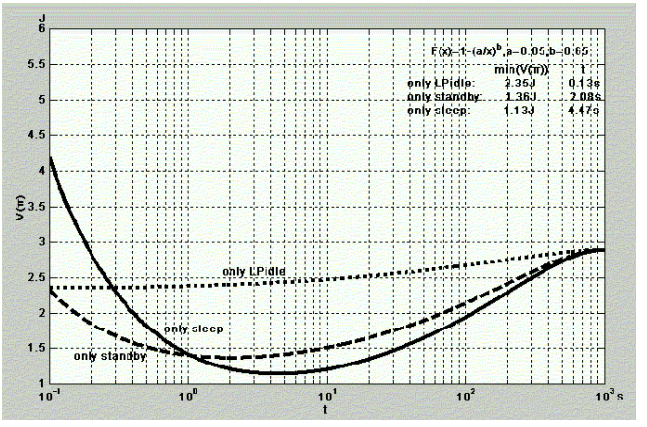


Fig. 3. The relationship between switch time and the total energy consumption in the Pareto distribution

1 Obviously, DPM is applicable in practice. Fig.3 shows that over 60% energy can be saved with correct use of the sleep mode.
2 DPM optimal strategy is quite stable. Fig.3 shows that the good effect can be received when the switch time is much later than some extremum value.
3 The exponential distribution supposition is not suitable to DPM. Fig.2 shows that there exists big error between the practical situation and this supposition.
4 The results of DPM optimal strategy change largely when the loads are different. Fig.3 shows much energy can be saved Standby and Sleep modes, but the LPidle mode is almost of no practical use for energy saving under the loads used in this experiment.

5 Conclusions

If service request arriving is taken as a stochastic event, timeout policy is enough for DMP. It is not necessary to search more complex DPM algorithm. The exponential distribution supposition taken by traditional queuing theory is not suitable to DPM. The reason that simple timeout policy works perfectly in practice is self-similarity.

References

1. Benini, L., Micheli, G.: Dynamic Power Management: Design Techniques and CAD Tools. Kluwer Academic Publishers, Norwell,MA (1997)
2. Douglis, F., Krishnan, P., Bershad, B.: Adaptive disk spin-down policies for mobile computers. In: The Second Usenix Symposium on Mobile and Location- Independent Computing (MOBLIC). (1995) 121–137
3. Helmbold, D., Long, D., Sherrod, B.: A dynamic disk spin-down technique for mobile computing. In: The Second Annual ACM International Conference on Mobile Computing and Networking. (1996)

4. Douglis, F., T.Killian: Adaptive modem connection lifetimes. In: The 1999 USENIX Annual Technical Conference. (1999) 27–41
5. Srivastava, M., Chandrakasan, A., Brodersen, R.: Predictive system shutdown and other architectural techniques for energy efficient programmable computation. IEEE Trans. on Very Large Scale Integration Systems **4** (1996) 42–55
6. Hwang, C., Wu, A.: A sredictive system shutdown method for energy saving of event-driven computation. In: IEEE/ACM International Conference on Computer-Aided Design. (1997) 28–32
7. Lu, Y., Micheli, G.: Adaptive hard disk power management on personal computers. In: IEEE Great Lakes Symposium on VLSI. (1999) 50–53
8. Benini, L., Bogliolo, A., Paleologo, G., Micheli, G.: Policy optimization for dynamic power management. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **18** (1999) 813–833
9. Qiu, Q., Pedram, M.: Dynamic power management based on continuous-time markov decision processes. In: The 36th ACM/IEEE conference on Design Automation Conference,. (1999) 555–561
10. Simunic, T.: Energy Efficient System Design and Utilization. PhD thesis, Stanford University, Stanford, CA 94305, USA (2001)
11. Chung, E., Benini, L., Bogliolo, A., Lu, Y., Micheli, G.: Dynamic power management for nonstationary service requests. IEEE Trans. on Computers **51** (2002) 1345–1360
12. WU, Q., ze XIONG, G.: A study of the optimal policy of dynamic power management for pervasive computing system. (to appear in Chinese Journal of Computers)
13. Crovella, M., Bestavros, A.: Self-similarity in world wide web traffic. evidence and possible causes. IEEE/ACM Trans. on Networking **5** (1997) 835–846
14. Taqqu, M.S., Willinger, W., Sherman, R.: Proof of a fundamental result in self-similar traffic modeling. ACM Computer Communications Review **27** (1997) 5–23
15. S.D. Gribble, G.S. Manku, D.R.: Self-similarity in file systems. In: The 1998 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS). (1998) 141–150
16. M.Harchol-Balter, A.Downey: Exploiting process lifetime distributions for dynamic load balancing. ACM Trans. on Computer Systems **15** (1997) 253–285
17. Harchol-Balter, M.: The effect of heavy- tailed job size distributions on computer system design. In: ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics. (1999)
18. Greenawalt, P.: Modeling power management for hard disks. In: The Conference on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. (1994) 62–66

An Adaptive Fault Tolerance Scheme for Applications on Real-Time Embedded System

Hongzhou Chen, Guochang Gu, and Yizun Guo

AUV Lab2, College of Computer Science and Technology,
Harbin Engineering University, Harbin HLJ 150001, China

Abstract. An adaptive fault tolerance scheme aiming at those real-time embedded systems that have single processor and limited resources is presented in this paper. As the fault behavior exhibited by system changes with time and environment, it can decide a suitable strategy among various reliable techniques that have been introduced into application software. Prediction on fault emergence is made by strategy decider according to its memory of fault emerging history. The application module is assigned to contact with fault tolerance controller through its stubs. Performance of the scheme is measured and evaluated in experiment. Some parameters are also discussed.

1 Introduction

Real-time embedded systems (RTES) have developed greatly during the past several decades. As these systems and the environment in which they work become more complex, there is a high likelihood that at any given time, some parts of the system will exhibit faulty behavior. The ability to tolerate this behavior must be an integral part of a real-time system. And it raises the requirement of designing reliable RTES for consumer with capability of fault tolerance (FT).

There are some fundamental FT models. N-modular Redundancy makes a number of identical copies of the software running on separate components, the majority decision is used among all the output of them [1]. This method, however, subjects to many redundant components and is thus luxurious for some systems not rich in hard and soft resources.

Recovery block approach combines checkpointing and backup alternatives to support recovery from failures [2]. All tasks are replicated but only one copy of each task is active at any time. A backup replica turns to run when the active one fails. Task may be completely restarted, which increases the chance of deadline missing, or else executed from its most recent checkpoint, which requires checkpoint updating and results in a large amount of overhead.

Since in some occasions, it is better to have less precise results on time than delayed precise results, forward error recovery comes into being [3], [4]. It is characterized by resuming from an erroneous state and making corrections that will clean up the damaged state, rather than rolling back and continuing from previously checkpointed state. But forward error recovery depends on accurate damage assessment and is often system-specific.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 474–480, 2005.

Different from static versions of these technologies mentioned above, adaptive FT (AFT) can operate in dynamic environments by deciding a suitable FT strategy for an arriving computation to assure required reliability. This feature thus has raised many AFT models, some of which focus on distributed system with multi-processor [5], [6], but too costly (if transformed) for single processor embedded system with poor resources; some try to provide scheduler with capability of FT [7], [8], which are more operating system-specific; most are mainly based on redundant hardware components [9], [10], and adaptive mechanisms such like continually attempting among alternatives to guarantee a new task in [9] entail lots of overheads.

The adaptive fault tolerance scheme (AFTS) presented in this paper targets those embedded real-time systems that possess single processor and poor resource. Its application-level FT achieved by way of incorporating reliability techniques within the application software is a powerful counterpart of lower-level FT when the latter does not provide enough required reliability by itself. Because detection and recovery of some faults are application-specific, implementation of them at the application level will work better.

Furthermore, AFTS features the adaptability of choosing suitable FT strategy for tasks when the emerging fault varies with the change of system-inside or system-outside environment. This capability is constructed on the predicting mechanism on fault emergence at current time according to its memory of fault emerging history.

The result of an actual project shows the good performance of AFTS, and some parameter settings are discussed.

2 Adaptive Fault Tolerance Scheme (AFTS)

Adaptive fault tolerance scheme has three compositions: some application modules (AM), fault tolerance controller (FTC), and replica management unit (RM), as Fig. 1 shows.

In the framework of AFTS, some FT stubs are inserted into application software to obtain the ability of detecting faults, sending fault tolerance request and recovering from where it encounters with a fault according to the response of fault tolerance controller. Fault tolerance controller contacts with application modules and replica management unit, decides an appropriate fault tolerance strategy when fault happens in running application task. The reliability of fault tolerance controller itself can be guaranteed by extra redundancy component if necessary or by aborative design of self-checking which maintains a state log.

Replica management unit collects all replicas of their counterpart primary AMs, attributes such like time/space constraint, addressing of every replica, are also implemented in RM, so as to cater to the retrieval by FTC. Actually, RM can be combined with FTC.

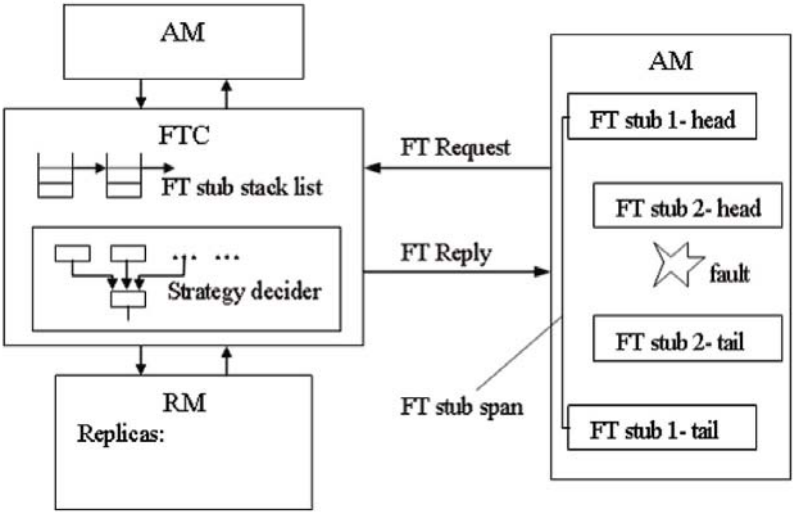


Fig. 1. The framework of adaptive fault tolerance scheme

2.1 Application Module (AM)

In AFTS, original application program code is equipped with some stubs (named FT stub) to take charge of fault detection and recovery via communication with FTC, and then turns to be the application module mentioned here. Since application's particular behavior and the situation in which it is located can be monitored effectively, it is convenient to handle problems at the locale, especially for some application-specific faults. The spot where fault occurs will be the perfect place where fault detection and recovery should be done.

Every FT stub comprises a head and a tail. They cooperate with FTC through two types of communication: FT request and FT reply, which are actually the messenger between a task's primary running (in AM) and redundant running (in FTC). The anterior includes two types of request:

1. H-type: ask FTC to register the key current state of corresponding stub span and raise an FT strategy, used at an FT stub head;
2. T-type: ask for result of the execution of FT strategy in FTC raised by relative H-type request, used at an FT stub tail.

Responsibility on reliability of an FT stub span, which means the section between head and tail of an FT stub, is consigned to its FT stub head and tail.

At every stub head, the stub sends an H-type request to FTC, and sets a timeout for fault detection. Either if it overtimes or any fault occurs in the execution procedure of the stub span before corresponding stub tail is arrived, a fault is identified, then go to stub tail. At corresponding stub tail, the stub sends a T-type request to FTC, when an FT reply is received, it begins to work on

acceptance testing and synchronization of results among primary running and redundant running.

FT stub span in AM is nestable, such as in Fig. 1. There is stub 2 in the span of stub 1, which forms a nested FT. The guaranteed sections in an AM are all of the stub spans, those sections outside any stub spans are provided with no FT, which run just as what their original versions do. So AM designer should be careful to determine into which section should a stub be inserted according to his or her experience of where faults often or maybe happen.

2.2 Fault Tolerance Controller (FTC)

Fault tolerance controller is the kernel part of AFTS, which communicates with application modules and replica management unit. It accepts and replies the request from AM, manages the memory of fault emerging, and selects replica from RM to carry out.

FT Stack Similar to the interrupt system in computer, an FT stub stack is kept for each application module to implement nested FT. Each item in a stack contains data such as program state, key variables, redundant results, etc., based on which communication between application's primary running and redundant running is undertaken.

When an H-type request comes, an item about its relative stub is pushed into relative stack, and an FT strategy is evoked and putted into effect subsequently; when a T-type request comes, an FT reply will take the redundant result to stub tail associated with the top item of relative stack, and then the top item is popped out. All the FT stacks of every AM are linked to be an FT stub stack list.

Memory of Fault Emerging Simulating the brain of mankind, FTC has memory of various faults emerging history in terms of fault emerging probability. It can increase relative memory quantity when a fault happens, and decrease when there is no fault. Let P_i denote the fault emerging probability of stub spans with the type of i, two stub spans with the same type mean that they have similar chance to potential fault. The memory of fault emerging changes with time according to equation (1):

$$P_i(t+n) = \text{Min}\ \{1, \rho_i P_i(t) + m\Delta\tau_i\} \tag{1}$$

where ρ denotes the factor of memory lapse, $0 < \rho < 1$. m means the count number of fault emergence in period t to $(t+n)$. $\Delta\tau_i$ is the memory quantity that should be enhanced when a fault emerges in a stub span with type of i.

Strategy Decider AFTS chooses three candidate redundant strategies for its decider: N-modular, recovery block and forward error recovery.

Because the target system owns a single processor, replicas of N-modular run concurrently rather than in parallel. Then N-modular and recovery block can share common replicas with each other. The difference is that replicas run concurrently in N-modular strategy but serially in the latter one. For the third strategy, lite versions are kept as its replicas running at situations where less precise but timely result is preferred.

When an item is pushed into FT stack, strategy decider begins to work. First, it evaluates the time and space requirement of the first two strategies. If none of them can meet the requirement of AM and system in term of time, the third strategy will be taken; otherwise, if there is only one that can meet the requirement of AM and system in term of time and space, that strategy will be chosen.

If there are two strategies that can meet the requirement of AM and system in term of time and space, strategy decider will decide the probability of fault emerging in relative stub span. If the estimated probability is less than the remembered one P_i associated with the stub span, N-modular strategy will be chosen, otherwise, recovery block will be chosen, since N-modular provides more reliability than recovery block. This means that N-modular strategy will be chosen with probability P_i.

After the strategy is determined, it is carried out immediately, and the result is recorded in FT stack. When a T-type request comes, corresponding results are sent to AM.

3 Experiments

Evaluation of AFTS is done in a multi-task system with the configuration: Mizilinux, S3C2410, 32 M memory, which serves as an unmanned monitor on city environment, dealing with sound and air data via communication with front-end and back-end. Measures are obtained by periodically inserting faults into the system.

As shown in Fig. 2(A), AFTS provides considerable reliability for system compared with original application. It is also found that although AFTS-8 or AFTS-2 has more replicas than AFTS-1, they perform alike. But when time cost is considered as shown in Fig. 2(B), AFTS-1 obviously excels others. This suggests that the strategy decider should choose a rational number of replicas, which is not the more the better.

Parameters ρ and τ in equation (1) are set to $\rho = 0.9$, $\tau = 0.1$. In experiments with various types of fault inserting, it is indicated that when fault emerging chance in a stub span with type of i is growing, ρ_i should become bigger and τ_i less. So application programmers should choose them according to different types of stub span.

4 Conclusions

This work presents an adaptive scheme for fault-tolerant system that possesses a single processor and limited resources. The application-level fault tolerance

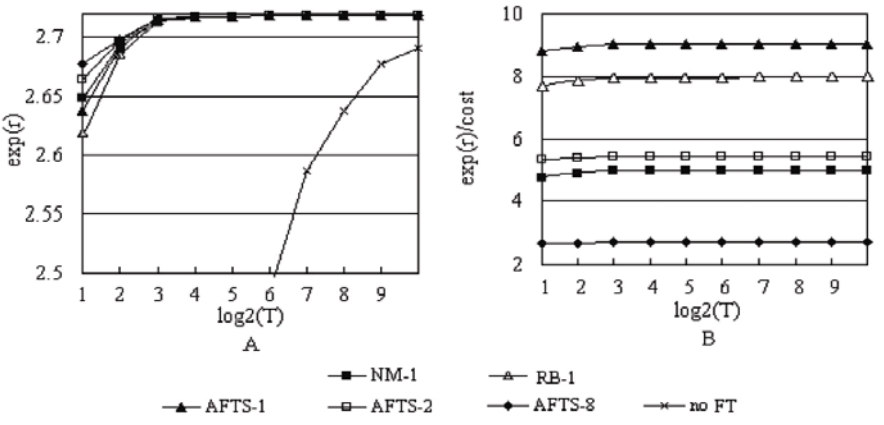


Fig. 2. Performance of no FT, AFTS and other FT with different degree of replication. *NM-1* and *RB-1* respectively refer to single N-modular and recovery block with one replica. *AFTS-n* refers to AFTS with n replicas for its N-modular or recovery block strategy. r denotes system reliability and T denotes the cycle of inserting fault. Part A shows e^r as a function of $log_2(T)$, part B shows a ratio (similar with the ratio of performance to cost) of e^r to time cost as a function of $log_2(T)$

makes it easy and effective to detect and recover from fault. Stubs inserted into AM take charge of fault detection and recovery and communicate with FTC conveniently. FT stack makes nested FT possible. Strategy decider chooses suitable alternative with change of environment. Memory of fault emerging provides basis for guessing mechanism. And the replica management unit maintains a collection of various replication of their primary AMs.

The experiment indicates that framework of AFTS is appropriate for poor system and simple to implement. It provides system with more reliability and less unbearable overheads than other FT strategies.

References

1. D. P. Siewiorek and R. S. Swarz: Relialble Systems Design and Evaluation. 2nd edn. Digital Press, Burlington, MA (1992)
2. J. J. Horning, H. C. Lauer, etc.: A Program Structure for Error Detection & Recovery. Lecture Notes in Computer Science, vol. 16 (1974) 172–187
3. J. W. S. Liu, W. Shih, etc.: Imprecise Computations. Proceedings of the IEEE, vol. 82, no. 1, Jan (1994) 83–93
4. David Kalinsky: Design Patterns for High Availability. Embedded System, vol. 15, no. 8 August (2002)
5. Fábio Favarim, Frank Siqueira, etc.: Adaptive Fault-Tolerant CORBA Components. Departament of Computer Science, Federal University of Santa Catarina (2003)

6. Olivier Marin, Pierre Sens, etc.: Towards Adaptive Fault-Tolerance for Distributed Multi-Agent Systems. In Proc. of ERSADS'2001, Bertinoro, Italy, May (2001)
7. LIU Huai, FEI Shu-Min: A Fault-Tolerant Scheduling Algorithm Based on EDF for Distributed Control Systems. Journal of Software, vol. 14, no. 8 (2003)
8. Larry Sieh, Peter Haniak, etc.: Implementing Transient Fault Tolerance in Embedded Real-Time Systems. US Army Tank-Automotive Research Development and Engineering Center, Warren, MI (2001)
9. O. Gonzalez, H. Shrikumar, etc.: Adaptive Fault-tolerance and Graceful Degradation under Dynamic Hard Real-time Scheduling. In Proc. IEEE Real-Time System Symp. (1997)
10. J. Haines, V. Lakamraju, etc.: Application-level Fault Tolerance as a Complement to System-level Fault Tolerance. The Journal of Supercomputing, vol. 16, (2000) 53–68
11. J. Karlsson, P. Folkesson, etc.: Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture. In 5th IFIP Working Conf. on Dependable Computing for Critical Applications, (1995) 267–287
12. O. Marin, M. Bertier and P. Sens: DARX - A Framework For The Fault-Tolerant Support Of Agent Software. ISSRE2003, Colorado, USA, November (2003)

Concurrent Garbage Collection Implementation in a Standard JVM for Real-Time Purposes

Yuqiang Xian, Ning Zhang, and Guangze Xiong

School of Computer Science & Engineering
University of Electronic Science & Technology of China
Chengdu 610054, China
xian@uestc.edu.cn

Abstract. Programming in Java is attractive for its maintainability and reliability, but much work is to be performed to apply it into the real-time software development. We try to address the two main issues, thread scheduling and garbage collection, with real-time concerns. Besides, the extra memory requirement caused by the introduction of garbage collection in systems is carefully concerned. As a result, a concurrent garbage collector that combines incremental collection and real-time scheduling is figured out and implemented in a standard JVM, which plays a key role in our future real-time JVM.

1 Introduction

The growing trend of larger scale embedded real-time applications caused by the evolution of the complexity of embedded systems requires a revolution in the developing methodology for such systems. Conventional developing methods that depend greatly on some popular yet traditional languages like C are inefficient and error-prone, which could be improved by the adoption of Java since it has been verified by enormous experiences in desktop and enterprise applications development that programming in Java alleviates the efforts of programmers and enhances the robustness and correctness of applications for its maintainability and reliability.

Reasons for Java in real-time have been presented, much work has to be performed to adapt it to real-time environments, though. A typical issue is the automatic memory management, say, garbage collection (GC) which introduces nondeterministic pausing time to user programs in the JVM, which is intolerable to real-time systems. We have proposed a concurrent garbage collection algorithm in previous work [1] with rigorous analysis and simulation verification. The results solve the problem well by not jeopardizing the original schedulability of the tasks with hard time constraint in the system. Other various approaches addressing the GC problem are also referenced in [1] .

Since many real-time systems are memory constrained, we further concerned the extra system memory requirement caused by the introduction of garbage collection in [1] and the resulting collection algorithm showed better performance

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 481–487, 2005.

on reducing the system memory requirement than any other concurrent collection algorithm by simulations, which is promising for the environment without strong memory.

We then provide an overview of our garbage collection algorithm in the following sections. Differing from our previous work, this paper concentrates more on the implementation of the algorithm that covers an incremental collector, a real-time scheduler, and real-time Java threads, etc. The implementation is based on Jikes RVM [2].

2 Review of the Algorithm

An incremental collector is the prerequisite of a concurrent collector. The system is composed by periodic Java threads (called "mutators" from the notation by Dijkstra et al. [3]) with hard time constraint in our model. They are scheduled using the Rate-Monotonic (RM) policy. The collection work, however, is serviced by a deferrable server (DS), and the server has a higher priority than any of the mutators. Therefore, through the classical DS algorithm that jointly schedules hard periodic and soft aperiodic tasks, the Java threads and collector run concurrently. We give the worst case response time of the collector by the following equation.

$$R_{\mathcal{G}} = C_{\mathcal{G}} + \left\lceil \frac{C_{\mathcal{G}}}{C_{ds}} \right\rceil (T_{ds} - C_{ds}) \tag{1}$$

where $R_{\mathcal{G}}, C_{\mathcal{G}}, C_{ds}, T_{ds}$ denote the worst case response time of the GC, the WCET of the GC, the capacity and the period of the DS respectively.

To avoid the danger of running the system out of memory, a certain number of memory resources should be reserved for the mutators when they are within a GC cycle. The minimum amount of reserved memory can thus be given by

$$M_{res} = \sum_{i=1}^{n} \kappa_i A_i \tag{2}$$

where A_i stands for the maximum amount of memory allocated by the mutator i (τ_i) during one period for it (T_i). κ_i is the maximum number of active instances of τ_i during $R_{\mathcal{G}}$ and is in direct ratio to $R_{\mathcal{G}}$.

In order not to jeopardize the schedulability of the mutators, the parameters of the DS (i.e. T_{ds}, C_{ds}) need to be assigned properly. A necessary and sufficient condition to guarantee the schedulability is given:

$$\text{for all } i \text{ between 1 and } n,$$
$$\left(\left\lceil \frac{T_i - C_{ds}}{T_{ds}} \right\rceil + 1 \right) C_{ds} + \sum_{j \in hep(i)} \left\lceil \frac{T_i}{T_j} \right\rceil C_j \leqslant D_i \tag{3}$$

where $hep(i)$ is the set of mutators whose priorities are not lower than τ_i.

Typically T_{ds} is set to the value of the shortest period among all mutators. Then C_{ds} is derived from the above inequality and the already known T_{ds}. But

from our analysis and verification in previous works, there's a better scheme on the parameter selection, which minimizes the worst case response time of the collector and the reserved memory requirement as well: T_{ds} is not fixed at the shortest period, but rather the final decision on the selection of the parameters is figured out by enumerating all the pairs of T_{ds} and C_{ds} that satisfy the inequality (3), which is possible to be done off-line since in our assumptions the parameters of the periodic mutators are known a priori, and the pair which makes $R_{\mathcal{G}}$ calculated by equation (1) reach a minimum is therefore our solution. It can be seen that the required reserved memory is minimized from equation (2), too.

3 Implementation

Jikes RVM [2] is adopted as the implementation platform and testbed for our garbage collection algorithm. It's a Java virtual machine written in Java and has been widely taken as the base of various researches in the VM field, profiting from the state-of-the-art VM techniques contained in it. (e.g. the just-in-time optimizing and adaptive compilers, an "efficient, composable, extensible and portable framework for building garbage collectors" [4], et al.) However, Jikes RVM is originally designed for servers and any real-time abilities were not concerned during its evolution. Therefore quite lot efforts are lying on the way to a real-time extension to the RVM, which is beyond the scope of this paper. However, the thread and memory management subsystems that are related to concurrent garbage collection are addressed in detail.

3.1 Thread Subsystem

The thread subsystem including the scheduling strategy in Jikes RVM lacks many real-time necessities, which prohibits the use of Java in real-time occasions. The lacks are listed below.

Insensitive to priorities. Although the interface that the VM presents to the users provides 10 priorities for the users to denote the preference of a sequence of threads for execution (see `Thread.java` in the Sun JDK or the GNU classpath library), the priorities are ignored in the internal processing of Jikes RVM. Furthermore, the original 10 priorities are not enough for the real-time processing.

Unpredictable scheduling. The threads in the RVM are distributed to one or more virtual processors, say, `VM_Processor`. The rule how the different threads are assigned to a particular virtual processor, and the order how the threads on the same virtual processor are placed in the queues (`readyQueue`, `transferQueue`, etc.) are both not deterministic. The scheduling between different virtual processors, however, depends totally on the underlying operating system. But it's not a case for a uni-processor system because there's typically only one virtual processor.

Absence of periodic thread specific operations. Since most tasks in a typical real-time system are periodic, convenient representations (e.g. the period, cost and deadline, etc.) and operations (e.g. the method doing the action of waiting for next period) of such tasks should be available. The RVM has nothing to show for those periodic thread specific operations.

Uninterruptible classes and methods. Many classes and methods in the RVM are uninterruptible and thread scheduling is disabled there. The typical examples are those of the garbage collector.

Each of the above issues is an obstacle to implement our algorithm and needs to be swept away. The Real-time Specification for Java (RTSJ) [5] presents a solution, which contains sufficient and convenient interfaces for real-time threads (priority extensions, real-time and even periodic thread specific operations, etc.) and for real-time scheduling. We adopt the same approach as that of the RTSJ in the thread and scheduling subsystem (excluding the `NoHeapRealtimeThread`), which offers the compatibility with the RTSJ as well. The listed shortcomings are thus corrected, e.g. as the priority of threads does make sense, the `readyQueue` and `transferQueue` for a virtual processor can be managed according to the priorities. To the uninterruptible classes and methods, currently we only remove those of the garbage collector by making it incremental; however, minimizing the occurrence of uninterruptible points in the total VM system requires a lot of hard work and it does make sense in future exploration. Furthermore, the deferrable server is treated as a periodic thread with the period T_{ds} and WCET C_{ds}. It's not ready to run at the beginning of each period as the normal ones, but instead, begins to run when there's collection work to do and C_{ds} is not exhausted. At last, the RVM must be built upon the option `VM.BuildForDeterministicThreadSwitching` enabled, which uses the count of method prologues executed rather than timer interrupts to drive preemptive thread switching, (in our case the count is set to 1, that is, thread switching is able to be performed just after one single uninterruptible operation finishes.), while in the default case, thread switching is enabled only at the discrete time points with an interval of 10 ms typically.

3.2 Memory Management Subsystem

Jikes RVM provides concurrent allocation and parallel collection of the objects. We modify the collection scheme by applying our concurrent algorithm. Because of the "stop-the-world" feature of the original collector, little or none synchronization between the user program and collector is necessary, whereas it's not the case for a concurrent collector because the user program (now can be called "mutators") can change the object reference graph during a collection cycle, which may cause incorrect collection if no additional mechanism appended. [3]

As we addressed before, our concurrent collector is a combination of an incremental garbage collector and a particular scheduling strategy, so a primary mission is to have the original collector in the RVM incremental. Fortunately, there're some commendable features in the memory management toolkit (MMTk) of the RVM such as the composability and extensibility [4] that make

a new collector implementation quite easy. Besides, the well organized internal data structures representing a Java object that distinguishes primitives from references are helpful for an accurate garbage collector, and the potential problems caused by arbitrarily switching between the user program and the collector are removed with the "safe points" generated automatically by the compiler. Both benefit an incremental collector. We take the mark-sweep plan of the MMTk as the basis of the incremental collector because it avoids the copying overhead and the semi-space cost of a copying collector, while fragmentation is kept at a low level owing to the segregated free lists provided in the RVM. [2]

Therefore, the remaining work to do will cover a modification to a write barrier (read barrier can be an alternative, but in most cases it introduces higher overhead.), some slight adjustment to each phase of a mark-sweep collector, and a trivial (or none) alteration to the internal object model.

Object model. We adopt the "tri-color" algorithm advocated by Dijkstra et al. [3] to implement an incremental collector, which marks the object with three different colors (i.e. black, grey and white) representing its different states. Therefore at least two bits of an object are required to store the color info, but the object model in the RVM only provides one bit for marking (another to denote large object) in the case when using a 10 bits hash code of the object in its header. Another scheme is to use the address of the object as its hash code, in which only two bits in the object header are required and thus the remaining 8 bits can be exploited for GC purpose. At present we are obliged to use the second hash code scheme for the tri-color marking. An alternative is to modify the object model so that it supports the tri-color marking in either case, which will cause correlative modifications to many other parts of the RVM, and we'll try it in future work.

Write barrier. The Dijkstra-style write barrier (the fine-grained solution) [3] for incremental garbage collection is introduced into the RVM. We override the method `writeBarrier` of class `BasePlan` to mark a white object grey if a black object attempts to reference it.

```
public final void writeBarrier(VM_Address src, VM_Address slot,
        VM_Address tgt, int mode)
        throws VM_PragmaInline, VM_PragmaInterruptible {
    VM_Magic.setMemoryAddress(slot, tgt);
    if (inMarkPhase() && hasScanned(src) && !isLive(tgt)) {
        traceObject(tgt);
    }
}
```

The barrier is called when the VM encounters the bytecode `putfield` or `putstatic`, through which the mutators' alterations to the reference graph during a collection cycle can't be missed by the collector and incorrect collection is thus avoided.

Collection phases. To handle the mutators' alterations to the reference graph, the collection phases should be modified slightly. For the marking phase, the original collector stores all the to-be-traced objects' addresses into some queues, and once an object has been scanned by the collector it is dequeued. However, we must also enqueue an object that has been marked grey by

mutators (through the write barrier), and the marking phase completes when there's no grey object, i.e. the queues are emptied. In the sweeping phase, the white objects (garbage) are appended to a free list that matches its size for future allocation, and the black ones are reset to white for next GC cycle. Some modifications to the preparing and releasing phases of GC that deals with the augmented color are also necessary.

So far we have presented the work to implement an incremental collector in the RVM, and the DS, as an ad hoc periodic real-time thread, is able to service the collection now, which forms a concurrent collector. At each yield point (i.e. when the thread switching attempt happens) the execution time of garbage collection is checked and if the time exceeds the maximum allowable time for current server period (i.e. the capacity of the server is exhausted), the collector thread yields into a `wakeupQueue` and is dequeued at the beginning of its next period. The implementation is like the code below.

```
if (elapsedMillis >= capaMillis) {  //whether the capacity was exhausted
    myThread.wakeupCycle = VM_Time.cycles()     //when should the collector be awakened
        + VM_Time.millisToCycles((double) (periodMillis - elapsedMillis));
    if (myThread.proxy != null) {
        myThread.proxy.wakeupCycle = myThread.wakeupCycle;
        VM_Scheduler.wakeupMutex.lock();
        VM_Scheduler.wakeupQueue.enqueue(myThread); //yield processor to the mutators
        VM_Scheduler.wakeupMutex.unlock();
    }
}
```

Although at present the collector is just a prototype and is less optimized, future work will perfect it to a good production.

4 Experimental Results

Our work in the implementation is in progress yet. Currently we have completed the prototype, and the correctness verification has been performed by executing the standard Java applications provided in the test harness of Jikes RVM and some RTSJ compliant real-time Java applications that have nothing to do with the aspects we have not considered up to now or we have discarded in the RTSJ. We achieved successful results.

The concurrency between the garbage collector thread and mutators is also tested and we select the results of running the program `LargeAlloc` that continuously consumes memory and turns it to junk quickly. The following example shows the execution behaviors of the collector ('+') and the mutator ('x') :

```
[cluer@clue gctest]$ rvm -verbose:gc -cp ./ LargeAlloc 500
LargeAlloc running with 500 Mb of allocation
Run with verbose GC on and make sure space accounting is not leaking

[Forced GC]+xxx[GC 1 Start 5.86 s   15348 KB +xxxxx+xxxx+xxxxx+xxxxx+xxxxx+xxxxx+xxx
xx+xxxxx+xxxxx+xxxxx+xxxx+xxxxx+xxxxx+xxxxx+xxxxx+xxxxx+xxxxx+xxxx+xxxxx+xxxx+xxxxx+
xxx+x+xx+x+xx+x+xxx++x+x+x+x+xx+xx+x+xxx+x+xxx++x+xxx+xx+x+xx+xx+x+xx++xx+xx++xx+x
++xx+x+xx+x+xxx+x+xx++x+xx+xx+x+xx+x+xx+x+x+x+x+xx+++x+xxx+xx+x+x++x+xxx+x+xxx+xxx+x
x+x+xxx+x+xxx+x+xxx+x+x+xx+xx++x++x+x+xx++xx+xx+xx++x++x+x+xx++xx+x+x+x+xx+x+x+xx++
x+x+x+xx++x+x+x+x+xx+xx++x+x++++x++
```

We can see from the results that the garbage collector is interrupted by the mutator even if it doesn't complete its work for current collection cycle (GC1), because it yields the processor to the mutator when it exhausts the capacity for one period of the deferrable server. The diversification of the number of 'x' between two neighboring '+' is mainly due to the fact that the mutator repeatedly requires memory allocation but the size of each allocation differs from that of the others.

Despite the fact that current GC implementation is a prototype, we have a coarse performance evaluation on it. The result shows that the average response time of GC increases dramatically (about 9 times longer than the original) caused by the removal of the uninterruptible points related to the collector, which reveals a general problem in general real-time systems: with more interruptible points for task preemption, the system is burdened with the large overhead introduced by frequently checking and performing thread switching. Therefore, design of a more efficient and more lightweight thread switching mechanism is essential to real-time systems.

5 Conclusions and Future Work

We have reviewed our concurrent garbage collection algorithm in this paper, and the emphasis is given to the implementation issues. Future work will complete a real-time extension by implementing the other aspects addressed in the RTSJ excluding the complex memory management scheme, such as synchronization (e.g. the priority inheritance algorithm) and asynchronous event handling mechanisms. Performance optimization should also be concerned, which includes the actions of minimizing the occurrence of the uninterruptible classes and methods, lowering the overhead caused by frequently checking and performing thread switching, and developing a real-time oriented JIT compiler.

References

1. Xian, Y., Xiong, G.: Minimizing memory requirement of real-time systems with concurrent garbage collector. ACM SIGPLAN Notices (2005, to appear)
2. Alpern, B., Attanasio, C.R., Barton, J.J., Burke, M.G., Cheng, P., Choi, J.D., Cocchi, A., Fink, S.J., Grove, D., Hind, M., Hummel, S.F., Lieber, D., Litvinov, V., Mergen, M.F., Ngo, T., Russell, J.R., Sarkar, V., Serrano, M.J., Shepherd, J.C., Smith, S.E., Sreedhar, V.C., Srinivasan, H., Whaley, J.: The Jalapeno virtual machine. IBM Systems Journal **39** (2000) 211–238
3. Dijkstra, E.W., Lamport, L., Martin, A.J., Scholten, C.S., Steffens, E.F.M.: On-the-fly garbage collection: An exercise in cooperation. Communications of the ACM **21** (1978) 965–975
4. Blackburn, S.M., Cheng, P., McKinley, K.S.: Oil and water? high performance garbage collection in Java with MMTk. In: The 26th International Conference on Software Engineering (ICSE'04). (2004) 137–146
5. Bollela, G., Gosling, J., Brosgol, B.M., Dibble, P., Furr, S., Hardin, D., Turnbull, M.: The Real-Time Specification for Java. Addison-Wesley (2002)

Relating FFTW and Split-Radix*

Oleg Kiselyov[1] and Walid Taha[2]

[1] Monterey, CA 93943. (oleg@okmij.org)
[2] Department of Computer Science, Rice University. (taha@rice.edu)

Abstract. Recent work showed that staging and abstract interpretation can be used to derive correct families of combinatorial circuits, and illustrated this technique with an in-depth analysis of the Fast Fourier Transform (FFT) for sizes 2^n. While the quality of the generated code was promising, it used more floating-point operations than the well-known FFTW codelets and split-radix algorithm. This paper shows that staging and abstract interpretation can in fact be used to produce circuits with the same number of floating-point operations as each of split-radix and FFTW. In addition, choosing between two standard implementations of complex multiplication produces results that match each of the two algorithms. Thus, we provide a constructive method for deriving the two distinct algorithms.

1 Introduction

Hardware description languages are primarily concerned with resource use. But except for very high-end applications, verifying the correctness of hardware systems can be prohibitively expensive. In contrast, software languages are primarily concerned with issues of expressivity, safety, clarity and maintainability. Software languages provide abstraction mechanisms such as higher-order functions, polymorphism, and general recursion. Such abstraction mechanisms can make designs more maintainable and reusable. They can also keep programs close to the mathematical definitions of the algorithms they implement, which helps ensure correctness. Hardware description languages such as VHDL [7] and Verilog [14] provide only limited support for such abstraction mechanisms. The growing interest in reconfigurable hardware invites us to consider the integration of the hardware and software worlds, and to consider how verification techniques from one world can be usefully applied in the other. Currently, programming reconfigurable hardware is hard [1]: First, software developers are typically not trained to design circuits. Second, specifying circuits by hand can be tedious, error prone, and difficult to maintain. The challenge in integrating both hardware and software worlds can be summarized by a key question:

> *How can we get the raw performance of hardware without giving up the expressivity and clarity of software?*

* Supported by NSF ITR-0113569 "Putting Multi-stage Annotations to Work" and Texas ATP 003604-0032-2003 "Advanced Languages Techniques for Device Drivers."

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 488–493, 2005.

Program generation [4,3] provides a seed for an answer to this question: it gives us the full power of a high-level language to generate descriptions of hardware (or any other kind of resource-bounded computation). Each generator represents a *family* of circuits. The practical benefit is a high-level of flexibility and reuse. The research challenge lies in finding analysis and verification techniques that can check and reason about this full family of circuits just by looking at the generator, and without having to generate all possible circuits. Resource-aware Programming (RAP) languages [12,5] are designed to address these problems by providing:

1. A highly expressive untyped substrate supporting features such as dynamic data-structures, modules, objects, and higher-order functions.
2. Constructs that allow the programmer to express the *stage distinction* between computation on the development platform and computation on the deployment platform.
3. Advanced static type systems to ensure that computations intended for execution on resource-bounded platforms are both type-safe and resource-bounded *without* generating all possible programs.

Developing a program generator in a RAP language proceeds as follows:

1. Implement the input-output behavior in an expressive, type-safe language such as OCaml [6]. For FFT, this step is just implementing the Cooley-Tukey recurrence.
2. Verify the correctness of the input-output behavior of this program. Because we used an expressive language, this step reduces to ensuring the faithful implementation of the textbook algorithm and the correct transformation of the program into a monadic style. The monadic transformation is well-defined and mechanizable [8].
3. Determine which parts of the computation can be done on the development platform, and which must be left to the deployment platform (cf. [11]).
4. Add staging annotations. In this step, staging constructs (hygienic quasi-quotations) ensure that this is done in a semantically transparent manner. Staging a program turns it into a program generator. A two-level type system understands that we are using quasi-quotations to generate programs (cf. [13]) and can guarantee that there are no inconsistent uses of first- and second-stage inputs. A RAP type system [12,5] goes further and can ensure that second-stage computations only use features and resources that are available on the target platform. The source code of the resulting *generator* is often a concise, minor variation on the result of the first step.
5. Use abstract interpretation techniques [2] to improve *generated code*, by shifting more computations from the generated code to the generator.

1.1 Contributions

The use of abstract interpretation in the RAP process (Step 5 in the above) was only recently proposed and investigated [5]. It provides a safe, systematic means for augmenting the generator with knowledge of domain-specific optimizations, and is a key difference between RAP and previous approaches to program generation including those used in the widely popular FFTW [3]. Despite the merits of this approach (see [5]), the

number of arithmetic operations in RAP generators for FFTW circuits only approached those in implementations generated by FFTW.

This paper shows that the technique of [5] can in fact be used to produce circuits with the same number of floating-point operations as in FFTW. In addition, choosing a different standard implementation of complex multiplication gives us circuits with the same number of floating-point operations as in split-radix FFT, another optimal FFT algorithm [9]. Thus, we provide a new, constructive method to deriving the two different classes of algorithms.

2 Matching FFTW and Split-radix

Previous work [5] uses staging and abstract interpretation to construct a concise generator of combinatorial FFT circuits. The starting point is the textbook decimation-in-time FFT $F(N,x)_k$ of the N-point complex-valued sample x_j

$$\begin{array}{ll} F(1,x)_0 & = x_0 \\ F(N,x)_k & = F(\frac{N}{2},E(x))_k + F(\frac{N}{2},O(x))_k \cdot w_N^k \\ F(N,x)_{k+\frac{N}{2}} & = F(\frac{N}{2},E(x))_k - F(\frac{N}{2},O(x))_k \cdot w_N^k \end{array} \text{ where } \begin{array}{l} E(x)_j = x_{2j} \\ O(x)_j = x_{2j+1} \end{array}$$

where $w_N^k = e^{-i2\pi k/N}$ is the Nth root of unity. E and O split the input x into even and odd parts, respectively. The transform is first applied to both halves, recursively, and the result is combined to yield the two halves of the transform sequence. Given the sample size N, the generator code literally follows this recursive algorithm. But instead of performing multiplications and additions, staging constructs [11] are used to generate a circuit that performs these computations. Observing that circuits generated this way are not always efficient, abstract interpretation is used to improve the quality of the generated code. In essence, abstract interpretation is used to enrich the generator with knowledge about several specific identities of real numbers, namely:

$$\begin{array}{rr} r \cdot 0 = 0 & sin(0) = 0 \\ r + 0 = r & cos(0) = 1 \\ r \cdot 1 = r & sin(\frac{\pi}{4}) = cos(\frac{\pi}{4}) \\ r + (-1 \cdot r') = r - r' & sin(t + \frac{\pi}{2}) = cos(t) \\ f \cdot r + f \cdot r' = f \cdot (r + r') & cos(t + \frac{\pi}{2}) = -sin(t) \end{array} \quad (1)$$

With these optimizations, the quality of the generated code in terms of number of addition and multiplication operations came very close to that of FFTW codelets [3]. But the resulting circuits had more floating point operations than in the corresponding FFTW codelets for sample sizes larger than 8. Whether it is possible to reach the same numbers as FFTW or other optimal algorithms remained open. This paper reports on three modifications to the abstract interpretation step that yield code with the same number as operations as FFTW:

1. Exploiting identities of complex roots of unity rather than their floating-point representations,

2. Switching from decimation in time (DiT) to decimation in frequency (DiF),
3. Exploiting the pattern of additions and subtractions in the algorithm.

FFT deals with complex-valued operations. Analysis of the algorithm shows that factors known at generation time are not arbitrary. In particular, they are never zero, and are always roots of unity ($e^{i2\pi j/n}$). If instead of representing such constants as floating points we represent them as rational numbers $\frac{j}{n}$, we can implement the identities on these values *exactly*. In particular, we are able to exploit the following identity:

$$e^{i2\pi j/n} \cdot e^{i2\pi j'/n'} = e^{i2\pi(\frac{j}{n}+\frac{j'}{n'})}$$

When roots of unity are represented as a pair of floating-point numbers, such equivalences do not hold except for trivial cases.

The decimation-in-frequency definition of FFT is as follows:

$$\begin{aligned} F(1,x)_0 &= x_0 \\ F(N,x)_{2k} &= F(\tfrac{N}{2}, E(x))_k \\ F(N,x)_{2k+1} &= F(\tfrac{N}{2}, O(x))_k \end{aligned} \quad \text{where} \quad \begin{aligned} E(x)_j &= x_j + x_{j+\frac{N}{2}}, \\ O(x)_j &= (x_j - x_{j+\frac{N}{2}}) \cdot w_N^j \end{aligned}$$

We have already noted that all complex multiplications in the FFT algorithm multiply a root of unity with a linear combination of input values. As a result, complex additions and complex subtractions in FFT always have the form $w_1 \cdot c \pm w_2 \cdot d$ where w_1 and w_2 are roots of unity. Such patterns can be computed in two different ways. The first is to do the multiplications by $w_1 \cdot c$ and $w_2 \cdot d$ first, and use the result for the final addition and the subtraction. The second approach is to re-write the expression as $w_1(c \pm w_2/w_1 \cdot d)$. The second approach is useful when the multiplication either by w_1 or by w_2/w_1 is trivial. The trivial multiplication is the one by ± 1, $\pm i$.

These are all the optimizations needed to match FFTW operation count.

2.1 Split-Radix FFT

Split-radix FFT is a particular FFT algorithm that aims to compute FFT with the least number of multiplications. In the general case, complex multiplication can be computed with four real multiplications and two real additions

$$(a + ib) \cdot (c + id) = (ac - bd) + i(ad + bc) \tag{2}$$

or, with three multiplications and five addition/subtractions

$$(a + ib) \cdot (c + id) = (t_1 - t_2) + i(t_1 + t_3) \text{ where } \begin{cases} t_1 = a(c + d) \\ t_2 = d(b + a) \\ t_3 = c(b - a) \end{cases} \tag{3}$$

The benefit of that particular formula among others with the same operation count is that when the factor $a + ib$ is known at generation time, two of the required additions/subtractions, namely, $b + a$ and $b - a$, can be computed at that time, leaving the other three additions and three multiplications to the run-time of the generated code.

Choosing equation (2) gives us the code that matches FFTW in operation count, whereas choosing equation (3) gives us the code that matches split-radix.

2.2 Experiments

The following table summarizes our measurements of the effect of abstract interpretation for FFT. The first column gives the size of the FFT input vector. The second column gives the number of floating-point multiplications/additions in the code resulting from direct staging. The column "RAP DiT" reproduces previous results [5]. The column, "RAP DiF 1" demonstrates the improvement of the more precise abstract interpretation described here. The next column shows the number of multiplications/additions in code generated by FFTW for the various problem sizes[3]. The column "RAP DiF 2" is "RAP DiF 1" but with complex multiplication with three real multiplies and five real additions, two of which are done at code generation time. The last column is the data for a split-radix algorithm with the complex input [9, Table II].

Size	Direct staging	RAP DiT	**RAP DiF 1**	FFTW [3]	**RAP DiF 2**	Split Radix
4	32/32	0/16	0/16	0/16	0/16	0/16
8	96/96	4/52	4/52	4/52	4/52	4/52
16	256/256	28/150	24/144	24/144	20/148	20/148
32	640/640	108/398	84/372	84/372	68/388	68/388
64	1536/1536	332/998	248/912	248/912	196/964	196/964
128	3584/3584	908/2406	660/2164	$\approx$ 752/2208	516/2308	516/2308
256	8192/8192	2316/5638	1656/5008	$\approx$ 2016/5184	1284/5380	1284/5380

We have used our FFT generator to generate all the circuit descriptions summarized by above table. To check the correctness of these implementations, we have translated them into C programs and checked their results and performance against FFTW. The following table shows the performance of the algorithms above as measured by the FFTW benchmark v3.1 on a Pentium IV 3GHz computer. The numbers show reported MFLOPS relative to FFTW for double-precision, complex, in-place, forward FFT. All code was compiled with GCC 3.2.2. The performance numbers show that on a Pentium IV, the floating-point multiplications are about just as fast as floating-point additions, and on modern super-scalar CPUs, the performance depends on many other factors (such as caching, pipelines stalls, etc.) rather than merely the floating-point performance. On DSP, FPGA and other similar circuits/processors, floating-point performance is usually the bottleneck.

Size	4	8	16	32	64	128	256
RAP DiF 1	335%	162%	97%	96.1%	83.1%	77.7%	68.8%
RAP DiF 2	323%	162%	102%	88.0%	79.2%	78.6%	69.6%
FFTW	100%	100%	100%	100%	100%	100%	100%

3 Conclusions

With systematic improvements to the domain-specific optimizations used, we found that staging and abstract interpretation can generate FFT circuits that match both FFTW and

[3] The numbers for FFTW are obtained from its codelets. FFTW does not have codelets for sample sizes 128 and 256. For those and larger sizes, FFTW uses the composition of smaller FFTW transforms. For those sample sizes, the operation counts in the table are estimates based on the counts for smaller sizes and on the Cooley-Tukey recurrences for the power-of-2 FFT algorithm.

the split-radix algorithm in terms of operation count. Furthermore, to generate circuits that match each of the two algorithms, all that is needed was to chose between two different definitions of complex multiplication.

Unlike FFTW, we know precisely where savings are coming from, and which particular equivalences contribute to which improvements in the code. We do not search for optimal code using extensive low-level optimizations at the level of real-valued terms. Rather, we use a small number of optimizations at the level of complex-numbers. Complex numbers are *the* domain-specific type for this application. We do not attempt to apply optimizations after generation, but rather, during generation. As such, our experience provides further evidence that abstract interpretation is a promising tool for expressing domain-specific optimizations in a program generation system.

Acknowledgements: We would like to thank Anthony Castanares, Emir Pašalić, and Abd Elhamid Taha for comments on this manuscript.

References

1. W. Boehm, J. Hammes, B. Draper, M. Chawathe, C. Ross, R. Rinker, and W. Najjar. Mapping a single assignment programming language to reconfigurable systems. In *Supercomputing*, number 21, pages 117–130, 2002.
2. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, pages 238–252. ACM, 1977.
3. Matteo Frigo. A fast Fourier transform compiler. In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 169–180, 1999.
4. C. S. Burrus I. W. Selesnick. Automatic generation of prime length FFT programs. In *IEEE Transactions on Signal Processing*, pages 14–24, Jan 1996.
5. Oleg Kiselyov, Kedar Swadi, and Walid Taha. A methodology for generating verified combinatorial circuits. In *the International Workshop on Embedded Software (EMSOFT 04)*, Lecture Notes in Computer Science, Pisa, Italy, 2004. Springer-Verlag. To appear.
6. Xavier Leroy. Objective Caml, 2000. Available from http://caml.inria.fr/ocaml/.
7. R. Lipsett, E. Marschner, and M. Shaded. VHDL - The Language. In *IEEE Design and Test of Computers*, pages 28–41, April 1986.
8. Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.
9. M.T.Heideman and C.S.Burrus. On the number of multiplications necessary to compute a length-2^n DFT. *IEEE Trans. ASSP*, ASSP-34(1):91–95, February 1986.
10. Oregon Graduate Institute Technical Reports. P.O. Box 91000, Portland, OR 97291-1000,USA. Available online from ftp://cse.ogi.edu/pub/tech-reports/README.html.
11. Walid Taha. *Multi-Stage Programming: Its Theory and Applications*. PhD thesis, Oregon Graduate Institute of Science and Technology, 1999. Available from [10].
12. Walid Taha, Stephan Ellner, and Hongwei Xi. Generating Imperative, Heap-Bounded Programs in a Functional Setting. In *Proceedings of the Third International Conference on Embedded Software*, Philadelphia, PA, October 2003.
13. Walid Taha and Michael Florentin Nielsen. Environment classifiers. In *The Symposium on Principles of Programming Languages (POPL '03)*, New Orleans, 2003.
14. Donald E. Thomas and Philip R. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publishers, 3rd edition, 1996.

Selecting a Scheduling Policy for Embedded Real-Time Monitor and Control Systems[1]

Qingxu Deng, Mingsong Lv, Ge Yu

School of Information Science and Engineering, Northeastern University
Shenyang 110004 China
dengqx@mail.neu.edu.cn, newlms@163.com, yuge@mail.neu.edu.cn

Abstract. An integrated solution to guarantee real-time requirements in embedded real-time monitor and control systems is presented in this paper. First a typical task model is abstracted from such a control system, together with specific characteristics of the tasks. These system characteristics lead to a hybrid scheduling policy of the rate monotonic algorithm and the sporadic server algorithm. The feasibility and rationality of such a policy are also analyzed. The rate monotonic algorithm is extended in this paper, providing a schedulability test that incorporates factors such as context switching overhead and task synchronization.

1 Introduction

Varies working environments lead to different type of embedded systems. Among these systems, a major one is industrial monitor and control system, which plays an important role in product lines. The background of this paper is such an industrial embedded system designed for real-time control of large-scale electro-mechanical devices.

In the reality, characteristics of different embedded systems may vary a lot. For example, major tasks in industrial control systems are hard real-time tasks, while key tasks of mass consumption systems often have soft real-time requirements. Currently there are mainly four different types of scheduling algorithms: static table driven algorithms, static priority based algorithms, dynamic priority based algorithms and bandwidth preserving algorithms. Since there is not an all-purpose scheduling policy applicable for every real-time system, the goal of this paper is to find a scheduling policy for our control system that can meet its specific real-time requirements.

The following sections are organized as follows. Section 2 presents a typical task model and the scheduling objectives. The reasons of using a hybrid policy are given in Section 3. An extended RM algorithm that takes resource sharing and system overhead into consideration is also given in this section. Section 4 gives a case study and conclusion remarks are given in Section 5.

[1] This work was supported by the National High Technology Research and Development Program of China (2002AA1z2308, 2002AA118030), and the National Natural Science Foundation of China (60473073).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 494-501, 2005.

2 Typical Task Model and Scheduling Objectives

The system is designed to serve high-pressure pumps for boilers. The core part of the system is abstracted into a task model, which is illustrated in Figure.1. Descriptions of tasks are illustrated in Table.1.

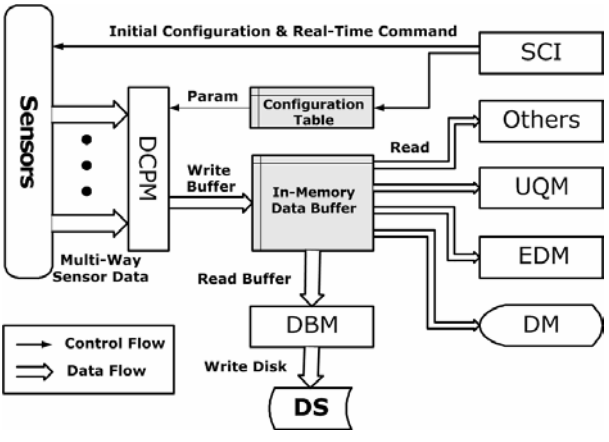


Fig. 1. A Typical task model of embedded real-time industrial control systems

Table 1. Introduction to major tasks in the model

Module	Full-Name	Functionality	RT Specs.	Arrival Pattern
DCPM	Data Collection & Processing Module	Process sensor data and store parsed data into IMDB	Hard	Periodic
DBM	Data Back-up Module	Back-up memory data to Disks periodically or when error occurs	Hard	Periodic
SCI	System Configuration Interface	Send commands to sensors and parameters to DCPM	Hard	Aperiodic
DM	Display Module	Display memory data in a friendly way to end users	Soft	Periodic
UQM	User Query Module	Provide users with querying interface for memory data	Soft	Aperiodic
EDM	Expert Diagnostic Module	A process to make deep analysis of in-memory data	Soft	Aperiodic

Considering the overall system, we can give some real-time requirements according to the controlled devices and task characteristics. First, system lifetime is one of the major aspects to characterize an embedded system. Our embedded industrial control system has a long lifetime. During this phase, hardware and software design are scarcely changed. Second, tasks in a real-time system can be sorted into hard or soft ones according to criticality, or classified into periodic or aperiodic ones by their arrival patterns. It can be seen from Table.1 that the system is composed of tasks with different real-time characteristics, and this may impose more

requirements on scheduling polices. Third, the tasks in this system are logically loose-coupling, but they are data-centrically coupled by sharing IMDB. Data sharing introduces priority inversion problems and synchronization blocking, so some mechanisms are required to guarantee data consistency and predictable system behaviors.

In summary, the embedded real-time control system has a complex task set composed of both control tasks and user applications. Data sharing plays a key role in this system. According to foregoing work, the following scheduling objectives can be given.

- *High Predictability:* In our system, all deadlines of hard real-time tasks should be guaranteed and high responsiveness should be provided to soft real-time tasks; predictable system behavior should also be guaranteed under transient overload. There is always a trade-off between high predictability and flexibility, and we choose to compromise flexibility to achieve high predictability.
- *Low Scheduling Overhead:* Embedded resources, such as computation capacity, memory and power supply, are still limited in embedded systems, although hardware technology has been greatly improved. This may impose limitations to the design of scheduling policy, so scheduling overhead should not be ignored.
- *Capability of Handling Hybrid Task Set:* Typical embedded control systems usually have hybrid tasks, which makes the system hard to handle. The scheduling policy should have the flexibility to deal with such a kind of task sets.
- *Incorporating Resource Sharing:* Ideal scheduling algorithms are developed assuming that tasks are independent of each other, and data sharing is a key characteristic of typical control systems. Resource sharing in priority based preemptive systems often introduces priority inversion and unpredictable blocking, which should be avoided to achieve higher predictability.

3 Scheduling Policy Selection and the Extended RM Algorithm

Up till now, there are many scheduling algorithms that are comprehensively studied. Our focus in this paper is priority-based preemptive algorithms, which are implemented in most real-time systems. Priority-based algorithms can be further classified into static or dynamic algorithms by considering whether priorities of tasks can be changed during the running time.

3.1 Motivations and Reasons to Select a Hybrid Solution

There is no single algorithm that can satisfy all the scheduling requirements since all the scheduling algorithms are designed for specific task sets with specific assumptions. This is the motivation to a hybrid policy. An integration of Rate Monotonic algorithm and Sporadic Server algorithm can meet the real-time scheduling requirements of our system.

The rate monotonic algorithm is selected from the following visions:

- *High Predictability:* High predictability is proved to be one of the superiorities of RM algorithm and a deterministic schedule can be calculated off-line. The internal mechanism guarantees predictable behavior even under transient overload.
- *Low Scheduling Overhead:* Compared to dynamic priority algorithms, RM has a very low scheduling overhead since priorities are assigned a prior. This property is often required as a key scheduling objective in embedded systems.
- *Incorporation of Resource Sharing:* Priority Ceiling Protocol, presented in [1], is widely used to handle resource-sharing problems in real-time systems. This protocol can be well incorporated into RM scheduler without too much alteration to schedulability test.
- *Engineering Simplicity:* There are extensive studies and implementations of RM algorithm in most embedded real-time operating systems. These experiences are quite helpful to our implementation in the view of engineering simplicity.

Sporadic Server, presented in [2], is selected for the following reasons:

- *High Responsiveness:* Compared with Polling Server and Deferrable Server, Sporadic Server can provide a higher responsive efficiency to sporadic tasks.
- *Guaranteeing Hard Deadline:* The Sporadic Server provides a uniform scheduling solution for both hard and soft sporadic tasks and deadlines of tasks with limited arriving frequencies can be guaranteed.
- *PCP Compatible:* If some conditions are met when integrating the two algorithms, Priority Ceiling Protocol can be adopted to resolve resource sharing problems under such a hybrid policy.

We choose this hybrid solution because, on one hand, the hybrid algorithm can meet the requirements that the system has on both periodic and sporadic tasks; and on the other hand, the Sporadic Server can be easily incorporated into the RM scheduler, since the Sporadic Server can be treated as an equivalently sized periodic task with a phase shifting. Sporadic Server can be considered as one of the extensions to RM algorithm.

3.2 The Extended Rate Monotonic Algorithm

Rate Monotonic algorithm [3] is advanced to schedule periodic hard real-time tasks. The key principle of this algorithm is that the priority of a task is assigned inversely to its period. The following assumptions should be satisfied: tasks are all periodic, deadlines are equal to periods, no dependencies among tasks, and run-time for each task is constant. RM schedulability test is given considering the worst-case situation when all tasks start simultaneously. We call the test FRMS in brief.

FRMS: Given a task set with n tasks, and each task is characterized by its worst-case execution time C and period T, the task set is schedulable using RM priority assignment if inequation (1) is satisfied.

$$\sum_{i=1}^{n} C_i / T_i \leq n(2^{1/n} - 1) \quad \textbf{(1)}$$

FRMS is a sufficient but not necessary schedulability test with low utilization threshold. In 1989, Lehoczky and Liu Sha advanced a sufficient and necessary schedulability test in [4]. We call it ERMS in brief.

ERMS: Given a task set with *n* tasks listed in priority descending order, define L_i in equation (2). The task set is schedulable if $\max_{\{1 \le i \le n\}} L_i \leqslant 1$. Here S_i referes to the scheduling points for the *i*th task.

$$L_i = \min_{\{t \in S_i\}} \sum_{j=1}^{i} C_j \cdot \lceil t/T_j \rceil / t \tag{2}$$

To extend the fundamental RM schedulability test, we first make an assumption that all tasks are independent of each other. In priority based preemptive systems, the PCP protocol [7] is a widely used policy, which is proved to be able to handle priority inversion, chained blocking and deadlock problems. The protocol works as follows: Each shared resource in the system is assigned a priority of the highest task that may access this resource. A job *J* starts a new critical section only if *J*'s priority is higher than all priority ceilings of all the semaphores currently locked by jobs other than *J*. The job *J* can be blocked by lower priority tasks at most once.

If B_i is defined as the longest blocked duration that task *i* may have in the worse case, then the FRMS test can be revised to inequation (3) and the definition of L_i in ERMS is changed to equation (4), hence resource sharing is taken into consideration.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \max\left(\frac{B_1}{T_1}, \ldots, \frac{B_{n-1}}{T_{n-1}}\right) \le n \cdot \left(2^{1/n} - 1\right) \tag{3}$$

$$L_i = \min_{\{t \in S_i\}} \left[\sum_{j=1}^{i} C_j \cdot \lceil t/T_j \rceil + B_i \right] / t \tag{4}$$

The assumption that context switching overhead is zero is also impractical and should be reconsidered. Since in our policy schedulability test is performed offline, we only focus on runtime overhead. Runtime overhead occurs when scheduling is activated. The implementation of the scheduling driven policy may lead to different scheduling activations.

In [10], Katcher presented the integrated interrupt event-driven scheduling, which is used in our system. The author also gave the building blocks of runtime overhead. C_{int} represents the time to handle an interrupt; the time to execute the scheduling code is C_{sched}; C_{store} and C_{load} is the time to save and load the TCB of a task; C_{trap} represents the time to handle the trap generated by normal completion of a task.

Katcher's analysis of scheduling activations is not enough since PCP protocol is not considered in his theory. The scheduler is activated in four situations: the release and termination of a task, and the blocking and wakening of a task with respect to resource sharing. The overheads are defined in equation (5), (6) and (7).

$$C_{preempt} = C_{int} + C_{sched} + C_{store} + C_{load} \tag{5}$$

$$C_{exit} = C_{trap} + C_{load} \tag{6}$$

$$C_{block} = C_{wake} = C_{sched} + C_{store} + C_{load} \quad (7)$$

During system runtime, there is no deterministic method to tell exactly how many times a low priority job may be preempted, so normal preemption overhead is added to the execution time of the *preempting* job. When considering blocking-triggered preemption, the PCP protocol guarantees that each task may experience at most once, so the overhead can also be calculated into the execution time of the preempting task. In the worst case, we define $C_i' = C_i + C_{preempt} + C_{exit} + C_{block} + C_{wake}$. If we replace C in FRMS and ERMS with C', then we get two more precise schedulability tests that can take context switching overhead into consideration.

The third extension to RM algorithm is the incorporation of Sporadic Server, since the algorithm enables the original RM scheduler to handle hard or soft sporadic tasks. Reasons why Sporadic Server is selected is given in Section 3.1. The Sporadic Server is intrinsically a bandwidth preserving algorithm since it preserves processor for sporadic tasks. The success of Sporadic Server is its specific replenishing policy. More results on Sporadic Server can be found in [2].

4 A Case Study

A case study is presented from the implementation of the system for one specific pump. The task set in this study contains 6 tasks and parameters of each task are illustrated in Table.2. For sporadic tasks, period means minimum inter-arrival time. Total worst-case context switching overhead for each task is 50ms.

Table 2. Real-time parameters of six major tasks

Task	Priority	Hard /Soft	Periodic /Sporadic	Period (s)	Computation Time(s)	
					Total	Shared Res.
T1	1	Hard	Periodic	2	0.4	0.05
T2	5	Hard	Periodic	60	1.0	0.12
T3	2	Hard	Sporadic	2	0.4	0.10
T4	3	Soft	Periodic	2	0.4	0.05
T5	4	Soft	Sporadic	10	0.5	0.12
T6	6	Soft	Periodic	60	2.9	0.12

The 6 tasks are assigned priority from 1 to 6 with the smallest number for the highest priority. IMDB is assigned highest priority according to PCP principles. Sporadic tasks can be treated as equivalent-sized periodic tasks when analyzing schedulability. A typical testing process may have four major steps.

Step 1: Add the task with the highest priority into the Scheduling Task Set;

Step 2: Test the schedulability of the task set with FRMS. If failed, execute Step 3; if all the tasks are in the task set, return TRUE; if not, go to Step 4;

Step 3: Test the Schedulability of the task set with ERMS. If failed, return FALSE; if all tasks are in the task set, return TRUE; if not, go to Step 4;

Step 4: Add another task with highest priority into the task set, and go to Step 2.

First assume that all soft real-time tasks do not exist, so the three hard real-time tasks can be scheduled with FRMS since inequation (8) holds.

$$\left(\frac{0.4+2\times0.05}{2}+\frac{1+2\times0.05}{60}+\frac{0.4+2\times0.05}{2}+\frac{0.1}{2}\right)\le 3\times\left(2^{1/3}-1\right) \quad (8)$$

If all soft real-time tasks are added, total system utilization is improved, making the less imprecise FRMS test yield failure. So we have to use ERMS to check the schedulability for the second round. Detailed process will not be presented, the result is that for all tasks from T1 to T6, the schedulable condition is satisfied at scheduling point 2, 2, 2, 4, 8, 22, respectively.

The deadlines of the six tasks are shown to be guaranteed according to the extended hybrid policy presented above. In some special conditions, period transformation technique can be adopted to raise the priority of a task, resolving the mapping of real criticality to priority. In conclusion, the RM algorithm guarantees that all hard real-time tasks meet their deadlines and the Sporadic Server algorithm can provide the shortest response time for soft real-time tasks.

5 Conclusion

This paper illustrates a whole process of guaranteeing the real-time requirements of an embedded real-time industrial control system from analysis of system characterizations to selection and extension of suitable scheduling policies. Representative control systems have specific real-time requirements, such as high predictability, low scheduling overhead and resource sharing. From previous work by other people, we conclude that a hybrid of RM algorithm and Sporadic Server algorithm with extended approaches is eligible for scheduling in our control system, and it's also helpful for selecting scheduling policies for similar systems.

References

1. Sha, L., Rajkumar, R., and Lehoczky, J.P.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, Vol.39, No.9. (1990)
2. Sprunt, B., Sha, L., and Lehoczky, J.P.: Aperiodic Task Scheduling for Hard-Real-Time Systems. Real-Time Systems, Vol.1, No.1. (1989)17–60
3. Liu, C.L. and Layland J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, Vol.20, No.1. (1973)46–61
4. Lehoczky, J.P., Sha, L., and Ding, Y.: The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. Proceedings of 10th Real-time Systems Symposium. (1989)166–171
5. Goossens, J., and Macq, C.: Performance Analysis of Various Scheduling Algorithms for Real-Time Systems Composed of Aperiodic and Periodic Tasks. The 5th International Conference on Information Systems, Analysis and Synthesis. (1999)
6. Jane, W.S. Liu.: Real-Time Systems. Pearson Education Press. (2002)
7. Ramamritham, K., and Stankovic, J.A.: Scheduling Algorithms and Operating Systems Support for Real-Time Systems. Proceedings of IEEE, Vol.82, No.1. (1994)55–67

8. Stankovic, J.A.: Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems. IEEE Computer, Vol.21, No.10. (1988)10–19
9. Stankovic, J.A., and Ramamritham, K.: What is Predictability for Real-Time Systems? Real-time Systems, Vol.2, No.4. (1990)247–254
10. Katcher, D. I., Arakawa, H., and Strosnider, J. K.: Engineering and analysis of fixed priority schedulers. IEEE Transactions on Software Engineering, Vol.19, No.9. (1993)

Sharing I/O in Strongly Partitioned Real-Time Systems

Ravi Shah[1], Yann-Hang Lee[1], and Daeyoung Kim[2]

[1]Computer Science and Engineering Dept.
Arizona State University, Tempe, AZ 85287
{ravi.shah, yhlee}@asu.edu
[2]School of Engineering
Information and Communications University, Daejeon, South Korea
kimd@icu.ac.kr

Abstract. Strongly partitioned real-time systems have been adopted to provide an integrated run time environment for applications with varied criticalities. This is achieved by guaranteeing spatial as well as temporal partitioning for the applications. To enable accesses to shared I/O devices in such an environment, this paper provides an effective model that the co-existence of any application does not hinder the execution of IO operations or spatial and temporal requirements of other applications. The model utilizes a microkernel-based approach to regulate the work of drive drivers and thus integrate the device-sharing model with the microkernel.

1 Introduction

Strongly partitioned real-time system (SP-RTS) supports co-existence of multiple execution partitions for various mission critical and time critical applications. All these integrated applications are guaranteed to meet their timing and space requirements while sharing resources with other applications [1, 2]

Partitioning the system is one of the common ways to provide such a guarantee. Partitioning aims at controlling the additional hazard created when a function shares its processor and resources with other functions. Traditionally, partitioning has meant separating the memory available in a system such that one partition cannot change or affect the state of other partitions. But real-time applications have their own timing requirements, which require broadening this common definition of partitioning to include time as a critical component of a partition and to have two different components [3, 4]:

- Temporal partitioning: to ensure that the service received from shared resources by the software in one partition is not affected by the ones in other partitions.
- Spatial partitioning: to ensure that software in one partition cannot change the software or private data of another partition nor command the private devices of other partitions.

Existing real-time OS such as WindRiver's vxWorks AE653 and Green Hill's Integrity-178B provide the platform to support partitioning for safety critical ARINC

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 502-507, 2005.

653 applications [5]. For instance, as shown in Figure 1, vxWorks AE653 includes a core OS based on vxWorks AE kernel which interacts directly with the computing platform (core module) and a partition OS based on vxWorks 5.5 Wind kernel. Partitioning support is provided inside the vxWorks AE kernel and a special private message-passing interface is available between the vxWorks 5.5 kernel (running in the partition) and the vxWorks AE kernel (running as the Core OS).

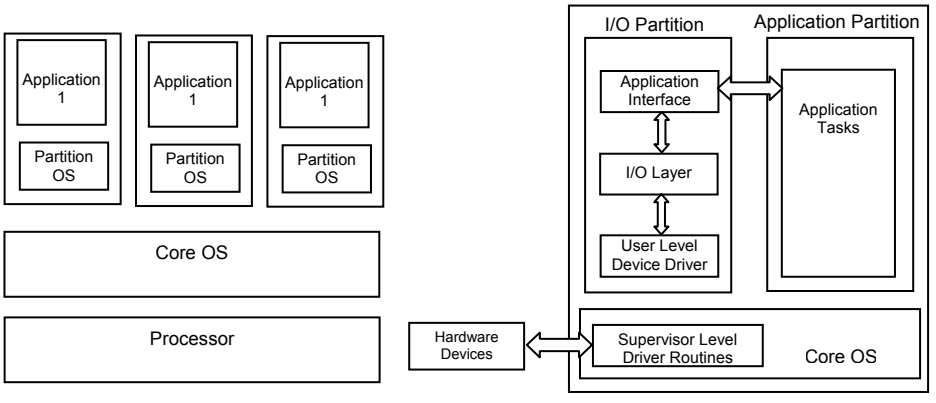


Fig. 1. vxWorks AE653 OS and Device Driver Models

This architecture involves a two level scheduling policy. A special partition layer is provided to serve the APEX (application executive) calls which requires interaction with the Core OS. Also, a fixed number of kernel threads are implemented to limit the number of blocking kernel calls by a partition at any given point of time [5].

vxWorks AE653 model implements a special I/O partition to accommodate various input/output devices. This partition is isolated from the application partitions and implements a user level device driver. Any application partition that wants to use an I/O device communicates to this partition using the ARINC ports. But the communication between the I/O partition the Core OS is done only in supervisor mode. This prevents any uncertified code from impacting the Core OS or application partitions, thus guaranteeing the spatial and temporal partitioning requirements of SP-RTS [5].

The multiple-layered kernel approach in [4, 5] is one of the most common approaches towards partitioning. It is preferred because of its clean design in terms of satisfying spatial and temporal partitioning requirements. This seemingly simple and elegant model seems to suffer some serious problems the moment we employ a partition to operate a shared hardware which is open to the outside world for interruption and that requires mediation from the CPU for its proper functioning. For instance, an invocation of IO operation can only be processed when the IO partition is scheduled. This results in a longer latency and limits IO bandwidth. Or on the other hand, if an IO partition is scheduled frequently, the context switch overhead may become significant.

In this paper, we propose a device driver model that can be implemented for SP-RTS models to allow the application partitions to share and access the devices. After presenting the design objectives for shared I/O devices in Section 2, the architecture

design and components are illustrated in Section 3. In Section 4, we show an experiment result in a simulated implementation. A short conclusion follows in Section 5.

2 Shared I/O Design Objectives

While dealing with shared I/O devices, the following issues need to be considered and solved:

1. *Authorized Access:* Not all partitions in the system might require access to the input/output devices. In this case, it is imperative that only the partitions that need to share particular devices are allowed to share those devices.
2. *Independent Functionality:* Despite the presence of shared devices on the system, the partitions should remain unaware of this fact, and should behave and perform as if no devices were shared.
3. *Spatial Partitioning:* Shared devices should not become a source of breaking the strong spatial partitioning of the system.
4. *Temporal Partitioning:* Any kind of time stealing of a partition from any other device being shared by other partition should be taken care for. That is, proper accounting of the time used by the shared devices should be done.
5. *Predictable timing constraints for I/O:* Despite the I/O paradigm (synchronous or asynchronous), it is imperative that proper timing guarantees are given to the partitions for finishing I/O operations [6]. Also, proper mechanisms should be in place so that the partitions can query device status.

3 Design of Shared I/O Architecture

The design of the system is based on the architecture of multiple-layer kernel such as SPIRIT [4] and vxWork AE653 [5]. The design tries to address all the issues that are introduced in SP-RTS due to the use of shared I/O devices in the system. Note that at microkernel level, partitions are scheduled by a cyclic scheduler. The partitions can have their own scheduling policy to schedule the tasks that run inside it. Following are the major components of the architecture as shown in Figure 2.

3.1 Publish-Subscribe Architecture

The microkernel supports a “publish-subscribe architecture” similar to the one used in [4] to help in controlling the access to the device by authorized partitions. The microkernel constructs a “publish table”, where it makes an entry for all the devices and device names that are registered for the device. On receiving a request from the partition to subscribe to a device, a memory device is created inside the partition, and an entry is made for this device in the subscription table.

3.2 Pseudo-device Driver

Pseudo device driver is a layer of software inside the partitions to support reading/writing or any normal device driver function into memory space rather than an actual physical device.

One memory device is added inside the partition for every shared device that the partition has subscribed to using the "Publish-Subscribe Architecture". The partition is aware of only the memory devices, i.e., it assumes the memory device to be the physical device and performs all the operations on it as if it were being done on the physical device.

The microkernel also uses this device driver layer to read the data that partition wants to transfer to the device and then perform the actual transfer of that data to the device. It is also used to provide the partitions with the data that is coming from the device for the partition. The pseudo device maintains multiple pointers inside it so as to perform all its tasks. Two circular queues are adopted to support the communication with the microkernel. The circular queues do not allow any overwrite of data in order to simulate the FIFO property generally present inside the devices.

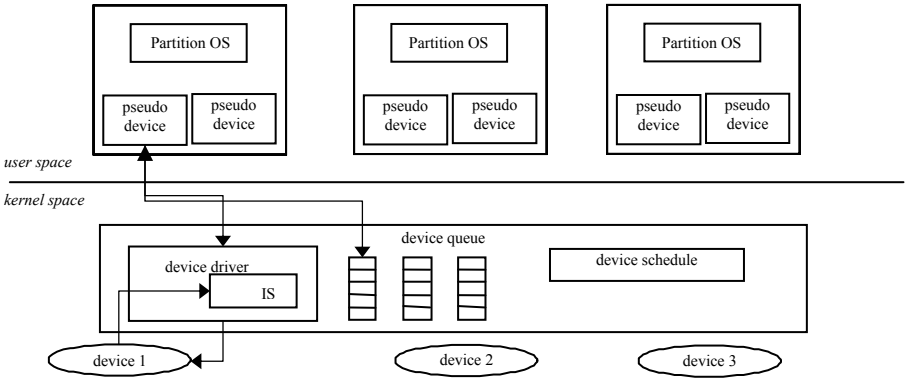


Fig. 2. Shared I/O Architecture in SP-RTS

3.3 Device Queues

Inside the microkernel, device queues (one per physical device) are maintained to act as a buffer for storing the data that is received from the device. These queues are assumed to be large enough to accommodate all the data that can arrive from the device before being transferred to the partition. The physical devices are configured in interrupt driven mode in the system. Once receiving any data, the device sends an interrupt to the CPU that is served through the device specific ISR.

Both these responsibilities are properly performed by the system using the device queues and low overhead ISRs. ISR simply transfers data from the physical device to the corresponding device queue. The time that it takes for the ISR to transfer the data from device to the device queue is accounted in the time of all the partitions that are

sharing this device. Though this mechanism introduces some overhead in terms of extra copying of the data, but it can minimize the time the ISR takes to transfer data.

3.4 Device Scheduler

It is a layer inside the microkernel that is responsible for transferring data from a pseudo device to its corresponding physical device, and from the device queues to the pseudo devices inside the partitions. The device scheduler is given its own time slices on the basis of IO bandwidth and scheduling policy that is defined at the system design time. The goal of this layer is to serve the application partitions and the devices in such a way that the partitions always get the requested IO bandwidth.

There are multiple ways in which the device scheduler can be scheduled. If the device scheduler is scheduled in partition context switch instants, then this type of scheduling is called "*interleaved scheduling*". It is desirable that the time for which the device scheduler is scheduling is directly proportional to the total buffer size of all the pseudo devices (corresponding to the shared physical devices) of preceding and succeeding partitions. This will give the device scheduler enough time to transfer all the data from the pseudo devices to the physical device, and also to transfer all the data from the device queues to the corresponding pseudo devices.

4 Experiment

The experiment is based on vxWorks RTOS that acts as supplement for the microkernel present in an actual SP-RTS. The prototype implementation therefore is a simulation of device sharing in SP-RTS. A cyclic scheduler is implemented as a task running on vxWorks, which schedules various partitions of the system. Also, device scheduler and device queues are implemented on vxWorks.

Along with the pseudo device drivers, the drivers for the physical device are also implemented. The reference implementation uses two device drivers - an IEEE 1284 parallel port device driver and a PCI based joystick device driver. The ISR routines of the device driver are written into the device queues as mentioned in section for device queues. While the parallel port can be configured in both read and write mode, the joystick can be configured only in read mode.

Readings are taken by varying different parameters like the number of partitions, number of devices, device bandwidth (i.e. the maximum amount of data that the device can transfer per unit of time). Figure 3 shows that given a fixed device scheduler time slice (i.e. total device bandwidth of all the shared devices), per device bandwidth decreases non-linearly with increasing number of shared devices in the system.

5 Conclusion

The proposed middleware based approach to share input/output devices can be a practical and efficient solution in SP-RTS architecture. The system design addresses

the two most important goals of SP-RTS while sharing the I/O devices. Spatial partitioning is achieved by introducing new components at pseudo-device layer and device scheduler layer. Temporal partitioning is accomplished by properly accounting the time taken for serving the I/O devices to the responsible partition. By considering the total bandwidth of all the pseudo devices that are created inside the partitions, time slice for device scheduling can be determined.

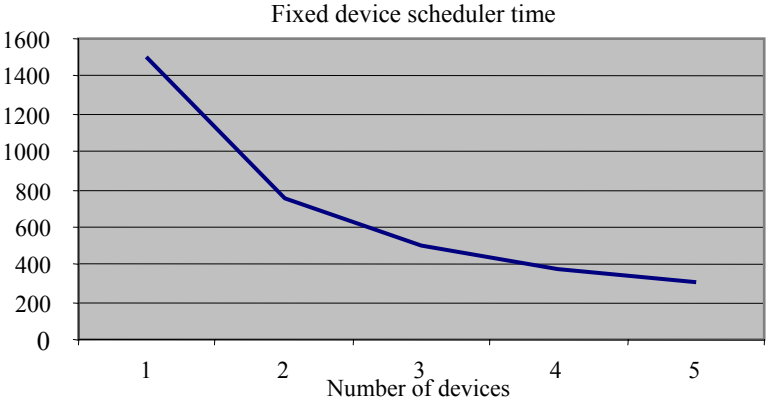


Fig. 3. Sharing of device bandwidth with multiple devices

References

1. Y. H. Lee, D. Kim, M. Younis, J. Zhou and J. McElroy, Resource scheduling in dependable integrated modular avionics, *Proceedings of IEEE International Conference on Dependable Systems and Networks*, June 2000.
2. Y. H. Lee, D. Kim, M. Younis and J. Zhou, Scheduling tool and algorithms for integrated modular avionics systems, *Proceedings of IEEE/AIAA Digital Avionics Systems Conference*, October 2000.
3. John Rushby, Partitioning in avionics architectures: requirements, mechanisms, and assurance, NASA Contractor Report CR-1999-209347, June 1999.
4. Y. H. Lee, D. Kim and M. Younis, SPIRIT-microkernel for strongly partitioned real-time systems, *Proceedings of the 7th IEEE Conference on Real-Time Computing Systems and Applications*, Pages: 73 - 80, November 2000.
5. Paul Parkinson, Safety-critical software development for integrated modular avionics, WindRiver White Paper, 2003.
6. Mark H. Klein and Thomas Ralya, An analysis of input/output paradigms for real-time systems, Technical Report, CMU/SEI-90-TR-19, July 1990.

The Efficient QoS Control in Distributed Real-Time Embedded Systems

You-wei Yuan[1,2], La-mei Yan[1], and Qing-ping Guo[2]

[1] Department of Computer Science and Technology, ZhuZhou Institute of Technology, (ZhuZhou, HuNan, 412008, China)
[2] School of Computer Science and Technology, Wuhan University of Technology, (Wuhan, 430063, China)
y.yw@163.com

Abstract. Developing distributed embedded systems based on Internet and Web technologies is a relative new trend in computer technologies. The paper shows the tree-layer client-server model in developing web-based distributed embedded systems. It also presents a model-driven approach for generating quality-of-service (QoS) adaptation in Distributed Real-time Embedded (DRE) Systems. Our control task model consists of the target task to be controlled, the adaptation task that implements the control algorithm. We have built a prototype of a middleware system which verifies our control model and shows the soundness of our approach. A detailed experimental analysis is also presented in this paper.

Key Words. Quality of Service (QoS); Real Time; Embedded Systems; Distributed Computation

1 Introduction

Day by day, we are witnessing a considerable increase in number and range of applications which entail the use of embedded computer systems. This increase is closely followed by the growth in complexity of applications controlled by embedded systems, often involving strict timing requirements, like in the case of safety-critical applications. Developing distributed embedded systems based on Internet and Web technologies is a relative new trend in computer technologies. It gives a common and well-known interface to the client (web browser) and a standard way for communication between devices.

Currently, embedded systems become more and more important and widely applied to everywhere around us, such as a mobile phone, a PDA, and an HDTV. [3]. In distributed real-time embedded systems, the communication time delay of remote method invocation has to be considered in real-time constrains. The underlying choice of transport mechanism (protocols and networking hardware) has a significant impact on overall system performance.

There exist many heuristic algorithms to do QoS adaptation utilizing buffering approaches [1], smoothing algorithms [2], acknowledgment-based approaches [3], pri-

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 508-513, 2005.

ority-based algorithms [4] and others. All these algorithms present possible system mechanisms, protocols, policies, hence partial solutions regarding how to deal with QoS adaptation, but they do not present a viable model and framework for QoS adaptation which allows to reason about QoS adaptation properties such as stability, agility and fairness, and to introduce new adaptation algorithms in an integrated and scalable fashion.

The paper shows the new tree-layer client-server model in developing web-based distributed embedded systems. This framework approach has also proven its cost-effectiveness in a number of industrial distributed embedded applications (primary substations for electricity transport, airfield lighting system), and is now being extended to incorporate inter-site distribution aspects. Power consumption has always been a primary design constraint for most wearable devices.

The rest of this paper is organized as follows. Section 2 presents three-layer client-server model in developing distributed embedded systems. Section 3 introduces the design of QoS adaptation in DRE systems. Section 4 describes the experiment results. The paper ends with some concluding remarks in section 5.

2 Three-Layer Client-Server Model in Developing Distributed Embedded Systems

With focus on current and future applications for the embedded web a central underlying principle is the Ethernet and TCP/IP protocol based communication.

The client-server model is the base model in developing internet-based information systems. In this model client processes request data from the server or there are two modules: client – the one that requests service, and server – the one that executes the service. Communication between the client and the server is based on the exchange of HTML (Hyper Text Markup Language) documents. Transportation of these documents is based on the protocol HTTP (Hyper Text Transfer Protocol). With focus on current and future applications for the embedded web a central underlying principle is the Ethernet and TCP/IP protocol based communication.

Figure 1 presents a conceptual framework of the proposed model. The object id, and request count of the most frequently accessed objects are in the router's memory. The actual objects reside in the server's cache. This approach is also related to Kohonen's elf-organizing feature map [5] technique, which facilitates mapping from higher to lower dimensional spaces so that the relationships among the inputs are mapped onto high reliable outputs.

The target task, utilizing the control theory, can have the following form:

$$\frac{dx(t)}{dt} \equiv \dot{x}(t) = f[x(t), u(t), w(t), t]. \tag{1}$$

$$z(t) = h[x(t), v(t), t]. \tag{2}$$

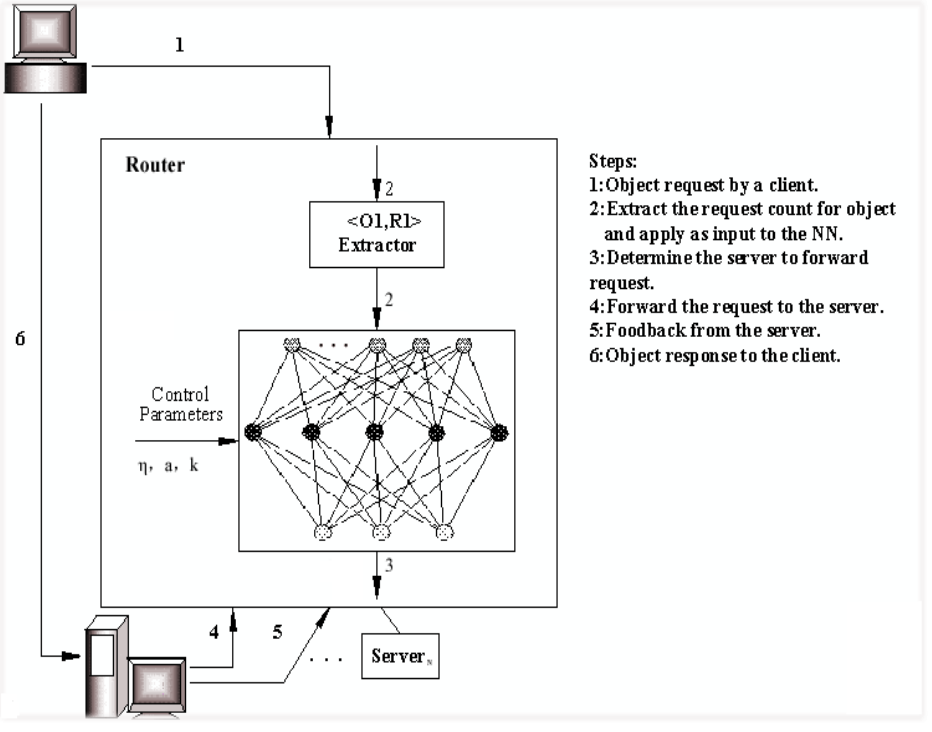


Fig. 1. A Conceptual Framework of our model

With the above definition, the target task is said to be at equilibrium when:

$$\dot{x} = 0 = f[x(t), u(t), w(t), t]. \tag{3}$$

where x denotes the vector of task states, u is the vector of controllable input parameters, z represents the vector of observed output parameters of the task, w is the uncontrollable variations in the task, and v denotes the observation errors.

The above stated definitions are generic and may be non-linear, time-varying, and too complex to derive a suitable control algorithm. Hence, in order to speed up and simplify the control, we will approximate the task control path by piecewise linear functions and work with target task model as follows:

$$x(k) = \Phi x(k-1) + \Gamma u(k-1) + w(k-1) \tag{4}$$

$$y(k) = Hx(k) \tag{5}$$

$$z(k) = y(k) + v(k) \tag{6}$$

3 The Design of QoS Adaptation in DRE Systems

Figure 2 shows QoS adaptation in Distributed Real-time Embedded Systems. The Lowest layer shows an abstract picture of a heterogeneous computing platform consisting of multiple devices connected over wire-line/wire-less communication links. The second layer shows the Architecture description of such a platform, and resource constraint description of the application requirements (such as power budget, real-time constraints etc.). The compiler (to be developed) takes the application functional specification, the ADL and RDL generate the services, and middleware configuration shown in the second highest layer, and their deployment information across the platform. [6]

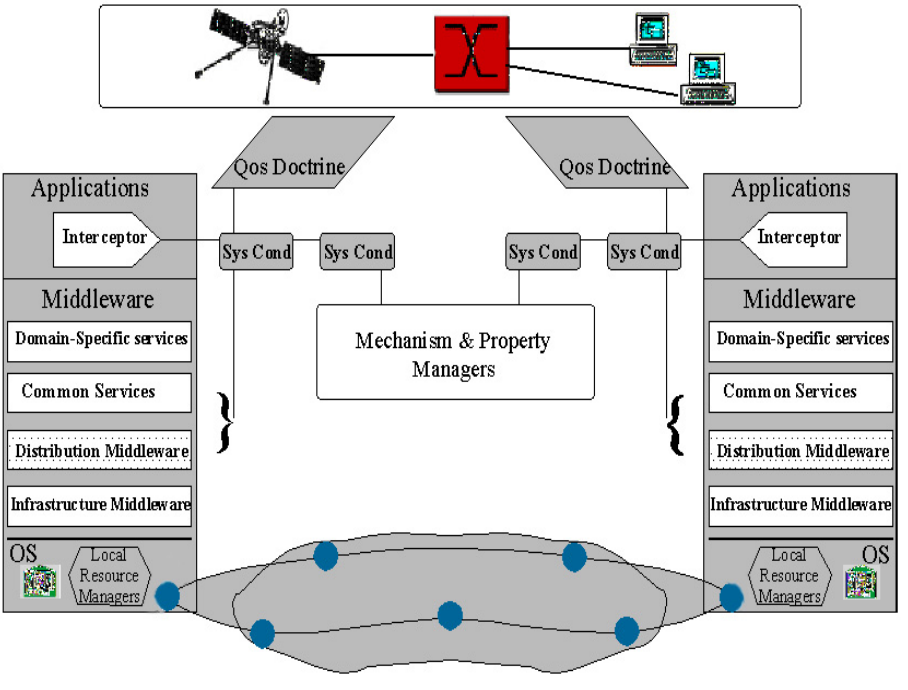


Fig. 2. QoS Adaptation In Distributed Real-time Embedded Systems

We design a hybrid control framework, consisting of a linear PID controller and a fuzzy controller. It is effective for maintaining the stability of the critical QoS parameter. This framework presents a feasible solution for QoS adaptation in distributed middleware systems.

Key features in this adaptation layer include (1) support for prioritized scheduling by partitioning requests for different QoS requirement into different threads and servicing these threads through different endpoints, (2) support for initializing endpoint

QoS properties, such as bandwidth reservation and flow pacing, and (3) support for portable scheduling control. Our concrete task control model for QoS adaptations must withstand not only minor fluctuations in shared resource availability, but also large changes.

4 Experimental Results

In this section, we present the results of our work. We discuss the performance of our algorithm for clustering. We compare performance of our algorithm with that of the K-Means clustering algorithm [1]. The tests are conducted using a Gateway PC with two 1.3GHZ P4 CPUs running Microsoft Windows 2000 and an Ultra- SPARC with four 300MHz UltraSparcs running SunOS 5.7. We compile the test on NT using Microsoft Visual Studio with Service Pack 3 and on Solaris using egcs version 2.91.60, but using full optimization. There are several things that make this a complex and challenging problem (i.e., the real-time requirements, resource constraints, and the distributed nature). Given these complex requirements, a QoS-enabled middleware solution has been proposed for this application [4]. Due to the highly dynamic nature of the application scenario, the adaptation of the QoS properties is mandatory. In this application, the goal of QoS adaptation is to minimize the latency on the video transmission. There are several ways of reducing the transmission rate: a) reduce the frame rate by dropping frames, b) reduce the image quality per frame, or c) reduce the frame size. In the example of this section, we consider dropping the frame rate only.

Figure. 3 shows: with no adaptation, almost all of the frames sent while the system is under load are lost. With a partial reservation and frame filtering, the middleware drops less important intermediate frames, but successfully delivers all full content frames (i.e., I-frames). With a full reservation, all frames are delivered successfully.

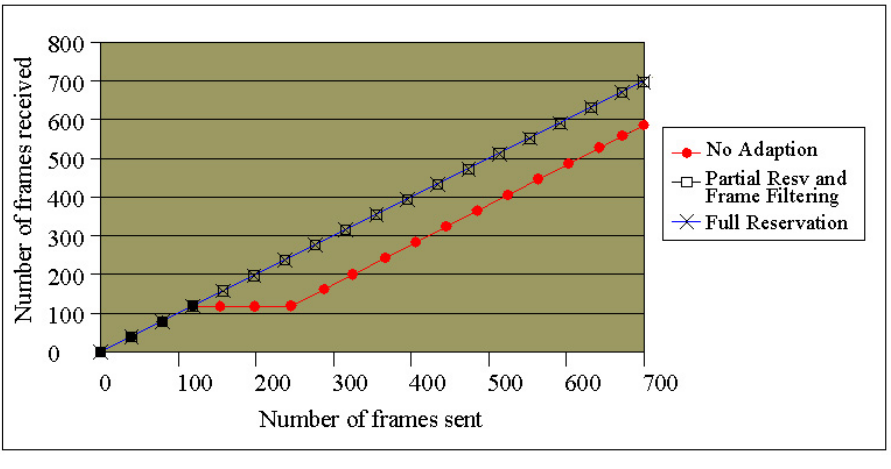


Fig. 3. Predictability of Image Delivery using Network Reservation

5 Conclusion

This paper presents an approach based on Model-Integrated Computing for simulating and generating QoS adaptation software for Distributed Real-time Embedded systems. As our work becomes more completely integrated with common middleware, which is complemented with appropriate resource management binding and scheduling services and policies, it enables a new generation of flexible DRE applications. The key focus of the approach is on raising the level of abstraction in representing QoS adaptation policies, and providing a control-centric design for the representation and analysis of the adaptation software. Using a model-based representation and employing generators to create low-level artifacts from the models, we have been successful in raising the level of abstraction, and providing better tool support for analyzing the adaptation software. At the same time, our approach results in increased productivity since we shorten the design, test, and iterate cycle by providing early simulation and analysis capabilities, and facilitate change maintenance as minimal changes in the models can make large and consistent changes in the low-level (CDL) specifications.

Acknowledgements

The work is supported by NSFC (Grant No: 60173046). And it is also supported by the science foundation of Hunan province ministry of finance and education. (No.04C717)

References

1. M. Shor, K. Li, and J. Walpole, “Application of Control Theory to Modeling and Analysis of Computer Systems,” in Proceedings of Japan-USA-Vietnam Workshop on Research and Education in Systems, 2000
2. G. Cao, W. Feng, and M. Singhal, “Online VBR Video Traffic Smoothing,” in Proceedings of 8th IEEE International Conference on Computer Communications and Networks, 1999, pp. 502–507
3. Z. Chen, S. Tan, R. Campbell, and Y. Li, “Real Time Video and Audio in the World Wide Web,” in Proceedings of Fourth International World Wide Web Conference, 1995
4. H. Chu and K. Nahrstedt, “CPU Service Classes for Multimedia Applications,” in Proceedings of IEEE International Conference on Multimedia Computing and Systems, 1999
5. T. Kohonen, “Self-Organization and Associative Memory”, 3rd ed. Berlin: Springer-Verlag, 1989
6. Radu Cornea1, Nikil. Dutt1, Rajesh Gupta2, Ingolf Krueger2, Alex Nicolau1, Doug Schmidt3, Sandeep Shukla4: FORGE: A Framework for Optimization of Distributed Embedded Systems Software. http://www.cs.wustl.edu/~schmidt/PDF/ipdps03.pdf

An Efficient Verification Method for Microprocessors Based on the Virtual Machine

Jianfeng An, Xiaoya Fan, Shengbing Zhang, and Danghui Wang

Northwestern Polytechnical University, China

Abstract. This paper presents an efficient verification method for microprocessors based on virtual machine. Under memory and I/O device models provided by the virtual machine, our simulation tool can not only simulate test programs but also operating systems. This simulation environment is close to the real environment of microprocessors, so it is sufficient for functional verification of microprocessors before tape out. At the same time, our simulation tool can automatically compare the simulation results using the virtual machine as reference model and find the error positions. This method takes full advantage of the virtual machine and greatly improves speed and efficiency of the verification procedure. This method has been successfully applied in the verification of an embedded microprocessor Amex86 designed in our laboratory for six months by five persons.

1 Introduction

As the development of high performance microprocessors, the computer architecture becomes much more complex. One of the most difficult problems designers face is how to verify the correctness of the designs. Using corner cases to test designs is a mostly manual process, where completion is hard to judge. Experience shows that the errors that are caught late in the design, many post-silicon, are interactions between different components in very improbable corner case situations [2]. Therefore, we must find suitable methods to ensure the correctness of a design.

The verification methods can be divided into two kinds: formal methods or simulation methods. For complex digital circuits, formal methods cannot be easily used because of the large state space. Therefore, simulation methods are often used in the engineering area [1]. In this paper, we use the method based on simulation. Many researchers do much work on this area. Some existent simulation tools include SimpleScalar [3], ASIM [4] and RSIM [5], etc. However, these simulation tools focus on the computer architecture research. They can only run the benchmark programs such as SPEC95/2000. This is sufficient for the performance evaluation, but insufficient for the microprocessor verification. For the microprocessor verification, the simulation tools must simulate as many programs in the real operational environment as possible, including the operating systems [6].

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 514–521, 2005.

Therefore, we must set up a simulation environment, which is close to the real environment of microprocessors. In this paper, we present a method based on virtual machine. The virtual machine can not only provide memory and I/O device models for simulating the operating systems, but also reference models for target design. The simulation tool can automatically compare simulation results using the virtual machine and find error positions. This method takes full advantage of the virtual machine and greatly improves speed and efficiency of the verification procedure. It has been successfully applied in the verification of an embedded microprocessor Amex86 in our laboratory.

Amex86 is an embedded microprocessor designed by Northwestern Polytechnical University. Its instruction sets are compatible with Intel 486 DX2 microprocessor [7]. In Amex86, there are more than 200 instructions. And every instruction's operand can have different choices: registers, immediate or memory; 8 bits, 16 bits or 32 bits operand size. If the operand is in memory, the address mode can also have different choices: direct mode, register indirect mode or others; 16 bits or 32 bits address size. Amex86 support three work modes: real mode, protected mode and virtual 8086 mode. Some instructions have different behavior under different mode. For the memory management, Amex86 supports segment and paging, which is managed by operating systems. Amex86 also supports internal exceptions and external interrupts. All of above functions should be verified carefully.

According to the function of Amex86, we divide the verification into two parts: instructions verification and system verification. The former emphasizes the verification of every instruction's function under different operands or different modes. The later emphasizes the verification of the whole architecture's function under operating systems, including memory management, exceptions and interrupts, etc. We use the verification method based on the virtual machine Simics [8] to accelerate the whole procedure and complete the whole verification for six months by five persons.

In the following part, Section 2 introduces the Amex86 verification based on the virtual machine Simics. Section 3 analyzes the verification coverage based on coverage rules. Section 4 draws a conclusion and points out the future work.

2 The Amex86 Verification Based on the Virtual Machine Simics

The key in the simulation verification is generation of test benches and check of the simulation results.

For the microprocessor verification, the test bench is equivalent with test instructions. Many researchers do much work to automatically generate test instructions. However, the instruction sets of Amex86 are very complex. The differences of instruction functions, instruction lengths and instruction codes is enormous. The auto generation of test instructions is very difficult.

According to the characteristics of Amex86's instruction sets, we can divide them into arithmetic and non-arithmetic instructions. For the former ones, we

must verify both the arithmetic result and EFLAGS, which is interrelated with the operands and executed sequentially. We use random generation methods to create possible operands to test. For the latter, non-instructions are not interrelated with operands and not executed sequentially. Manual generations of test benches are necessary.

2.1 Random Generation of Arithmetic Instructions Based on Templates

Random generation of arithmetic instructions is based on templates. Templates contain the information of instruction format and possible operands. For example, part of the MOV instruction's template is as following:

```
//MOV
//Register to Register/Memory
1000000 w mod reg r/m
//Immed to register (Short format)
1011 w reg immed
```

Every template starts with a new line (// implies a remark line). Templates can use numbers to present an instruction's code. Templates can also use signs to present a variable operand. In the upper example, w means 0 or 1 and reg means any number between 000 and 111 in binary.

The instruction generator will identify the variable part in the template and generate a random value. All possible values could be generated as long as the simulation time is long enough.

2.2 Manual Generation of Non-arithmetic Instructions

The non-arithmetic instructions, such as the control transfer instructions and operating system instructions, cannot be generated automatically, because these instructions need special execution environment and particular execution sequences. However, these instructions are not interrelated with the values of the operands. So we can generate all the test instructions and write the corresponding test benches.

Both arithmetic instructions and non-arithmetic instructions are compiled into standard EXE format in the MASM 611 compiler for the subsequent simulation and verification.

2.3 Auto Check of Simulation Results

Because the test benches are very long, we must find some methods to check the simulation result automatically. The best way is to compare Amex86 with a functional model, such as Simics [8], Bochs [12], etc. We select Simics because Simics provides plenty of API functions and flexible interfaces. We use socket

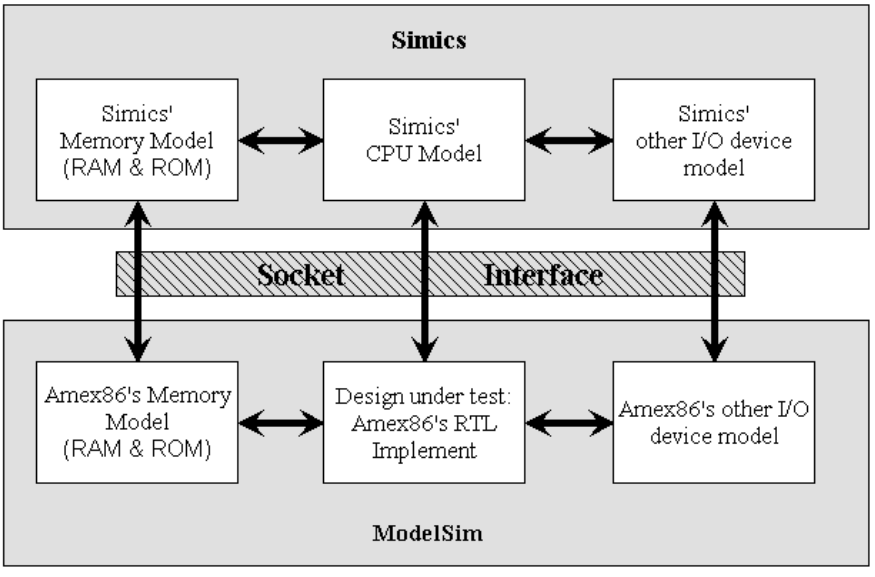


Fig. 1. Automatically check simulation results with Simics

as the communication method between the simulation tool ModelSim and the virtual machine Simics, as shown in Fig 1.

As a whole, the simulation environment is composed of the simulation tool and the virtual machine. The simulation tool can simulate the Amex86 VHDL codes. The virtual machine can provide the memory and I/O models and reference models. The whole simulation procedure is as following:

- 1.Put the EXE-format test program into DOS operating system installed in the Simics.
- 2.Start Simics and Listen on the socket port 21.
- 3.Start ModelSim and connect on the socket port 21 with Simics.
- 4.Input simulation parameters, such as simulation start address, end address and compare sizes etc.
- 5.Run Simics to the start address.
- 6.Synchronize ModelSim with Simics, including Amex86 inner registers and Amex86 outer memory.
- 7.Let ModelSim run instructions specified by the compare sizes.
- 8.Let Simics also run instructions specified by the compare sizes.
- 9.Compare the simulation results between ModelSim and Simics and write the result into the log file.
- 10.If there are severe errors (for example, EIP is different), goto 6.
- 11.If there are trivial errors (for example, EAX is different), force the correct content in the ModelSim.

– 12.If simulation arrives at the end address, simulation will complete. Otherwise, goto 7.

The user can easily find errors in the log file and resume the error point to observe the error conditions. A screen snapshot captured in the simulation is shown in Fig 2. The Simics console in the top left corner can observe all states in the virtual machine. The Simics output window in the down left corner can observe the screen output. The ModelSim console in the top right corner can observe the log file and find the time when errors happen. The ModelSim waveform window in the down right corner can show the states in ModelSim. The user can analyze the error causation with these information.

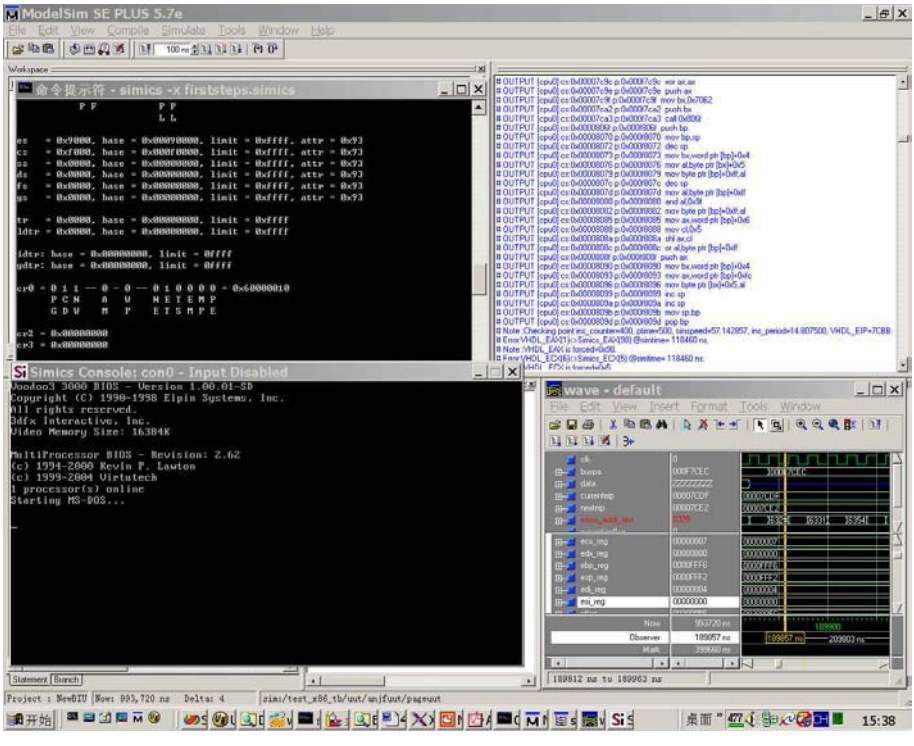


Fig. 2. A simulation snap using ModelSim with Simics

The simulation speed and corresponding simulation environment are shown in Figure 3 and Table 1. We can see the relation between the simulation speed and the compare size is nearly linear. At the earlier state in the verification, we should use compare size 1 to compare every instruction's execution result. After most errors have been removed, we can increase the compare size to increase the simulation speed.

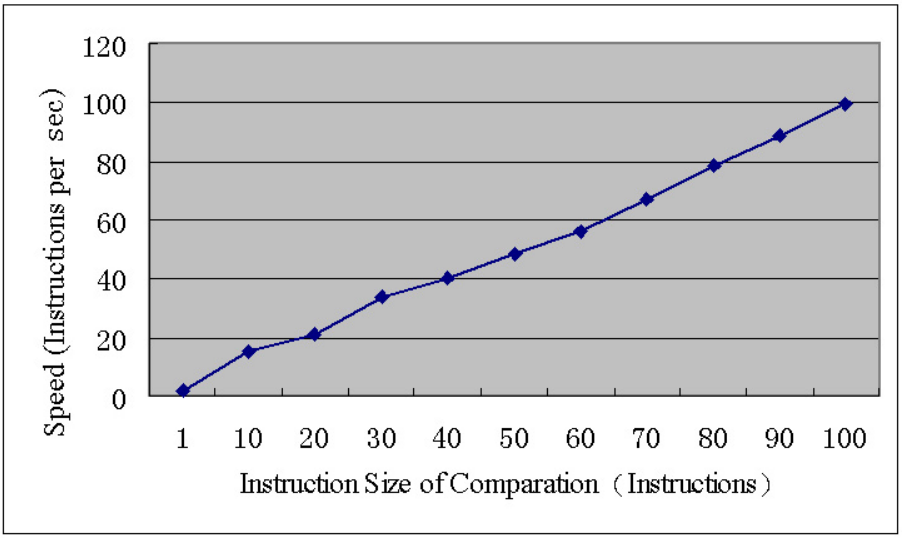


Fig. 3. The simulation speed comparing with different instruction sizes

Table 1. The simulation environment

Item	Configuration
PC	Intel Pentium 4, Windows 2000 Professional, 1 G DDR
Simics	2.0.8
ModelSim	5.7e

3 The Amex86 Verification Coverage Analyze Based on Coverage Rules

The chip scale is becoming larger and larger, so simulation of all possible inputs is impossible and coverage analyze based on coverage rules is needed. For Amex86, we use coverage analyze based on coverage rules to check if the verification is complete.

According to the characteristics of Amex86, every instruction is decoded into one or several microinstructions. So Amex86 can be regarded as a combination of microinstructions and execution units. The coverage analyze should also include microinstructions and execution units. ModelSim [9] can provide detail coverage about the HDL, such as line coverage and condition coverage etc. So we can use ModelSim to provide functional coverage of execution units.

However, ModelSim cannot count the execute time of every microinstruction. We add some statistical code in the Amex86 test bench to collect this information. The result is written into a text file which is formatted as following:

0000:Y:1-2-3
0001:N:5-6-7-4001-4002-4003-8
... ...

The number indicates the microinstruction's address, and Y or N indicates the microinstruction is tested or not.

Using this method, we can get the coverage of the microinstructions and execute units. We can add additional test benches according to the coverage analyze result until both microinstructions coverage and lines coverage are 100%.

4 Conclusions and Future Work

Through the method in the paper, we finish all the verification work for six months by five persons. As a result, we design a main board for the Amex86 microprocessor, whose interface is compatible with the PC104 system. We put Amex86 and its mainland into an Xilinx Virtex2 xc2v8000 device. And we can run DOS operating system steadily on the FPGA platform. Therefore, the verification result for Amex86 based on the virtual machine is successful. After the final synthesis and layout, the Amex86 microprocessor will be taped out using SMIC 0.18 um in this year.

This method is easily used for verification of other microprocessors' design, such as PowerPC, ARM, etc. We will continue to use it on another microprocessor, Longtium R2, designed in our laboratory, whose instruction sets are compatible with PowerPC 750. And We plan to use the Synopsys Vera tool to collect the functional coverage and guide the verification procedure. This will make the verification work more efficient.

5 Acknowledgement

We thank Shangang Zhang for discussions on the ideas in the paper. This work is supported by the National Defense Pre-Research Project of the "Tenth Five-Year-Plan" of China under Grant No. 41308010307. We also thank the Virtutech AB for Simics.

References

1. Cui Guangzuo etc.: System level simulation, emulation and debug method for processors -a new method based HW /SW. Computer Research and Development, China (2001)
2. Richard C. Ho, C. Han Yang, Mark A. Horowitz, and David L. Dill: Architecture Validation for Processors. In the Proceedings of the 22nd International Symposium on Computer Architecture, Santa Margherita Ligure, Italy (1995)
3. D. Burger and T. Austin: The SimpleScalar Tool Set, Version 2.0. University of Wisconsin Computer Sciences Technical Report 1342 (1997)

4. Joel Emer,Pritpal Ahuja,Eric Borch,Artur Klauser,Chi-Keung,Luk Srilatha,Manne Shubhendu S.,Mukherjee,Harish Patil,Steven Wallace,Nathan Binkert,Ann Arbor Roger Espasa Toni Juan: Asim: A Performance Model Framework. IEEE Computer (2002)
5. V. Pai, P. Ranganathan, and S. Adve: RSIM Reference Manual Version 1.0. Technical Report 9705, Department of Electrical and Computer Engineering, Rice University (1997)
6. Jian shen: Effective techniques for processor validation and test, PhD thesis. University of TEXAS at AUSTION (1999)
7. Intel: Embedded Intel 486 processor family developer's manual. http://www.intel.com (1997)
8. Simics: http://www.simics.net/
9. Model Technology: ModelSim Foreign Language Interface. http://www.model.com/
10. Altera: Quartus II Device Handbook. http://www.altera.com/
11. Anthony J.Massa: The debug method and skill using ROM monitors. http://www.eetchina.com/art_8800312705_617681,617693.htm (2003)
12. Bochs: http://bochs.sourceforge.net/
13. Carl J. Mauer, Mark D. Hill and David A. Wood: Full-System Timing-First Simulation. The 2002 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems (2002)
14. Joon-Sco Yim,Chang-Jae Park, Woo-Seung Yang: Verification Methodology of Compatible Microprocessors. The proceedings of 34th Design Automation Conference (1997)
15. Junqiang Hu, Jinsheng Li, Peilin Hong: The FPGA verification based ModelSim FLI interface. Application of electronic technique Vol.28 No.7, China (2002)
16. John Morris: Reconfigurable Logic: A Saviour for Experimental Computer Architecture Research. The 8th Asia-Pacific Computer Systems Architecture Conference (2003)
17. Christopher Giese etc.: Protected mode demo code. http://www.execpc.com/ geezer/os/ (1998)

EFSM-based Testing Strategy for APIs Test of Embedded OS

SongXia Hao, XiChang Zhong, and Yun Wang

Hopen Software Engineering Co.,Ltd.
No.4 Southern 4th St, ZhongGuanCun, P.O.Box 2717 Haidian District,
Beijing 100080, P.R. China
{sxhao,ywang}@hopen.com.cn
xczhong@sec.ac.cn

Abstract. In this paper, we present a two-phase approach to generate test data for the EFSM model of an embedded operating system. We first build the EFSM model of the system. The model can then be used to automatically generate test data for testing of APIs. We also point out certain issues that need further study.

1 Introduction

With advanced computer technology, systems are getting larger to fulfill more complicated tasks, but at the same time, they are also becoming less reliable. While many software test techniques are eminently sensible and extremely useful, they often have no real theoretical basis. This has encouraged the view that software testing is an imprecise process that is more of an art than a science [1]. The most promising approach is using both formal methods and software testing. In recent years a new consensus has developed that formal methods and testing are seen as complementary.

Because of the constraints on the amount of time and effort that can be spent in testing, the automation of test derivation becomes a very important issue. In this paper, we also present a new algorithm to generate the test data automatically from our model.

The rest of the paper is organized as follows. In Section 2, basic concepts and notions are introduced, and the EFSM model is presented. Section 3 discusses the feasible path problem, and describes the algorithm for automatic test data generation. We conclude in Section 4 by discussing possible future topics.

2 Formalization of Hopen APIs

Because of the complexity of systems, it is often proposed to use the formal techniques to support the automation of the testing. Among the formal description techniques, a shared drawback is the inadequacy to take into account systems of

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 522-527, 2005.

industrial size, where data are widely used. Thereafter, research studies have been based on the models extended with data and related concepts such as parameters, variables and operation on data. In this paper, we choose the EFSM (Extended FSM), a very powerful model for verification and test derivation [2].

2.1 Basic Concepts

We first define all the objects of an EFSM, and introduce the necessary notations.

Definition 2.1. An *extended* finite state machine (EFSM) M is a 4-tuple

$$M = (I, O, S, T) . \quad (1)$$

where I, O, S, and T are finite sets of input symbols, output symbols, states, and transitions between states from S respectively, such that each transition $t \in T$ is a 6-tuple (s, x, P, op, y, s'), where

- s, $s' \in S$ are the initial(current) and final(next) states of the transition, respectively;
- $x \in I$ is input, I is the set of all possible inputs, and more formally I is the set of all input vectors, D_{xi} is the domain of input variable x_i;
- $y \in O$ is output, O is the set of all possible output vectors y = (d_1, d_2,…,d_m) such that $d_i \in D_{yo}$ where D_{yo} is the domain of output variable y_o;
- P, op are functions, defined over input vectors, namely,
  P: $Dxi \rightarrow$ {True, False} is a predicate;
  op: $Dxi \rightarrow Dyo$ is an output vector function.

We present some conventions to simplify the notations for transitions. Specifically, we normally use (s—x,P/op, $y \rightarrow s'$) to denote a transition $t \in T$. If, in t, P is a True constant, P can be dropped from the transition. At the same time, the output parameter function can only be absent when output y has no output parameters at all.

2.2 EFSM Model

Hopen OS [3] is an RTOS running applied in many applications. We will use it as an example to discuss our approach. Due to the space, we use the event module for illustration, which is shown in Fig. 1, though the system is modeled as a collection of EFSMs. Events in Hopen OS represent one synchronization mechanism. Tasks waiting for events move to the ready state if any requested event is received. Events can only be cleared by the tasks to which they are assigned.

In Fig.1, our working example has 4 states and 37 transitions. The input is the 5 API names of the event module, and the output is the return value of the function (*retval*). The *retval* has one parameter *errno* (error number provided by the function). S = {s_0,s_1,s_2,s_3}, where s_0 is the initial state, s_1 is an event with no task waiting, s_2 is an event with waiting task, s_3 is a destroyed event with waiting task. The input set includes all the API functions [4] of event module, where I = {CreateEvent, DestroyEvent, OpenEvent, WaitEvent, SetEvent} (we use CE, DE, OE, WE, SE for short). Transitions with the input CE are represented as t_{11},t_{12},…,t_{18}, DE as t_{21},…,t_{26}, and so on as the sequence in set I. One may notice that t_{61} is denoted as a dashed line

in Fig.1, which means that all the event calls finish and return to the initial state. The descriptions of the transitions are listed in Appendix A.

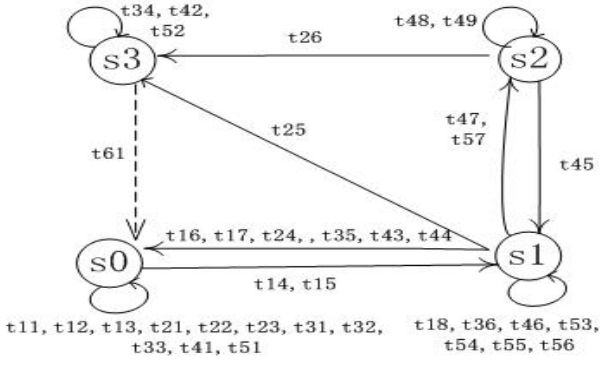


Fig.1. EFSM model of Hopen

3 Test Generation

Testing consists of a number of execution scenarios of a system implementation against a selected set of test cases, called a test suite. In this section, we will describe an algorithm to automatically generate test data [5] for the traversal of a given path in the model in section 2.2.

3.1 Reachability Analysis

In EFSM testing, not all the generated paths are feasible. For some subsequences of the paths may not be executable because the transition enabling predicates (also called constraints) along the path cannot be satisfied with any inputs. Reachability analysis could eliminate the problem, which will result in the reachable tree of the model.

The reachable tree of an EFSM is a tree showing the behavior of the machine starting from all possible initial states under all possible input sequences. For every input sequence, the tree contains a path starting from the root, and every node is annotated with the corresponding current (and/or initial) transition.

The reachable tree is also a directed graph, and thus graph theoretic concepts and algorithms can be used in the analysis of EFSM's. For example, we may want to visit the nodes (transitions), record the order of the visit, and explore the structure of the graph such as its connectivity properties. This can be done by a depth-first search (DFS) or breadth-first search (BFS).

To make the analysis, we first introduce some definitions.

Definition 3.1 A (specific) path [5] is a sequence of nodes $p = <p_1, p_2,\ldots, p_n>$, where p_n is the last node of path p. Whenever the execution of P(x) traverses a path p, we say that x traverses p. A path is (absolutely) feasible if there exists an input $x \in S$ that traverses the path.

The EFSM in Fig.1 has 55 paths. Based on the above definition, path (t_{14},t_{35}), (t_{15},t_{17}),and the paths begin with (t_{15},t_{36}) are infeasible. Fig.2. shows the reachable tree.

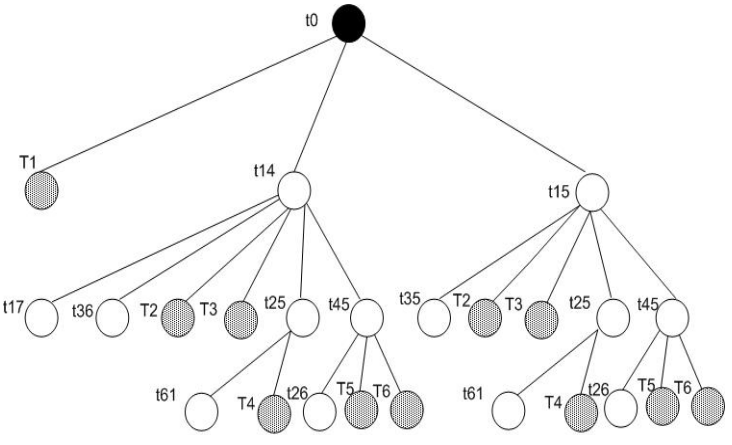


Fig. 2. Reachable Tree of Fig.1

In this diagram, nodes represent the transitions in the EFSM. The black solid circle is the initial node. We combine some nodes to make reduction, and represent them by a grey solid circle. They are T_1={t_{11},t_{12},t_{13},t_{21},t_{31},t_{32},t_{33},t_{41},t_{51}}; T_2= {t_{16},t_{24},t_{43},t_{44}};T_3={t_{18},t_{46},t_{53},t_{54},t_{55},t_{56}};T_4={t_{22},t_{34},t_{42},t_{52}};T_5={t_{47},t_{57}};T_6={t_{48},t_{49}}.

3.2 Test Generation Algorithm

Typically we wish to produce a set of input sequences among which satisfy some test criterion. To satisfy a criterion, we often generate a set of paths from the EFSM first, and then generate test data for each path.

When testing an EFSM, the extended data portion has to be tested also to determine the behaviors of the implementation. That means both control and data flow techniques should be combined in the EFSM testing. From this opinion, the traditional methods for testing FSM are no longer adequate. We now give a detailed algorithm for generating the test data for execution of a given path in the reachable tree.

It is an iterative algorithm. In each iteration, we:

Step1: Finding all the feasible paths in the reachable tree.

Step2: If the transition sequence of one path is the sub-path of another, remove this path. Each selected path corresponds to an explicit test purpose. Then, group the paths by test purpose.

Step3: To form an independent and executable test case, it is necessary to append some statements (e.g. the judgment of test verdict). All the test cases make up the test suite.

Some transitions in Fig.2 can be partitioned by their data attribute. For example, t_{33} and t_{53} include two cases when event handle={-1,32}, while t_{44}, t_{54} include three cases

when event handle={0,1,2}. Applying the algorithm discussed above, 59 test cases are generated. According to their actions on event, we classify them into 5 groups (the first five lists in Table 1), and the others are in the group Other.

Table 1. Test Suite of Event Module

Group	*Creation*	*Destroy*	*Opening*	*Waiting*	*Setting*	*Other*
Number of Cases	8	7	8	19	15	2

To illustrate the algorithm, we select the path $\{t_0,t_{14},t_{17}\}$ in Fig.2 for demonstration. It is a feasible path, so go to step 2. The test purpose of this path is to verify the error handling when creating an existed event. In step 3, add the verdict judgment.

The test case generated from the path $\{t_0,t_{14},t_{36}\}$ is shown below.

1) Call CreateEvent with a specific key parameter, if fail, go to 6);
2) Call OpenEvent with the same key value with 1), if fail, go to 6);
3) Call DestroyEvent to destroy the event created in 1);
4) Output the verdict PASS and return;
5) *Output the verdict FAIL and return.*

The algorithm is actually the DFS (Depth-first search) of the reachable tree. So the space our algorithm needs is only b*m nodes, where b is the branch factor of the reachable tree, and m is the maximal depth. The search time complexity is O(b^m).

4 Conclusion

In our two-phase approach, we first build the EFSM model of the system, and then use the model to automatically generate test data for testing of APIs. Although we use Hopen OS as an example, our approach is efficient and provides a practical solution to generate test data for APIs test in any embedded OS. Further research is needed to elaborate testing strategies combining the algorithm with various coverage criteria for specifications in the form of other existing formal description techniques.

Acknowledgements

The authors thank Ms. Yue Gao for carefully reading the manuscript and providing insightful and constructive comments.

References

1. Jonathan P. Bowen, Kirill Bogdanov, eds. IEEE Computer Society Press, (2002) 91-101.
2. Petrenko A., Boroday S., and Groz R. Confirming configurations in EFSM. Proceeding of FORTE XII (1999).
3. Xi-Chang Zhong, Ni Zhang, Embedded Software and Hopen System, BUAA press (2004).

4. Hopen 3.0 C library and API Manual, http://www.hopen.com.cn.
5. J. Edvardsson. A survey on automatic test data generation. ECSEL,(October 1999) 21-28..
6. B. Korel, Automated Software Test Data Generation, IEEE Transactions on Software Engineering, Vol.16, No. 8, (1990) 870-879.

Appendix A: Transition Description Table of Fig. 1.

Name	Description
t11,t21,t31, t41,t51	(s0—x, invalid arguments /errno = EINVAL, -1→s1), x = {CE,DE,OE,WE,SE}
t12	(s0—CE, no enough memory/errno = ENOMEM, -1→s0)
t13,t32	(s0—x, no file handle/errno = ENOMEM, -1→s0), x = {CE, OE }
t14,t18	(S—CE, valid arguments, handle→s1) ,S={B,E}
t15	(s0—CE, valid arguments, handle→s1)
t16	(s1—CE, too much ipc objects created/errno = EMFILE, -1→s1)
t17	(s1—CE, event existed/errno = EEXIST, -1→s1)
t22,t52	(s3—SE, event destroyed /errno = EIO, -1→s3), x = {DE,SE }
t23	(s0—DE, not event handle/errno = ENOTTY, -1→ s0)
t24,t43,t53	(s1—x, invalid file handle/errno = EBADF, -1→S), x = {DE,WE,SE }, S={B,B,E}
t25	(s1—DE, valid arguments, 0→s3)
t26	(s2—DE, destroyed , NULL→s3)
t33	(s0—OE, event not exist/errno = ENOTTY, -1→s0)
t34	(s3—OE, valid arguments/errno = ENOTTY, -1→s3)
t35	(s1—OE, event without name /errno = EINVAL, -1→s1)
t36	(s1—OE, valid arguments, handle→s0)
t42	(s3—OE, valid arguments/errno = EBADF, -1→s3)
t44,t54	(s1—x, not event handle/errno = ENOTTY, -1→S), x = {WE,SE },S={B,E}
t45	(s2—WE, timeout≠0, NULL→s1)
t46	(s1—WE, time=0 /errno = EAGAIN, -1→s1)
t47	(s1—WE, time out/errno = ETIME, -1→s2)
t48	(s2—WE, timeout <0,NULL→s2)
t49	(s2—WE, with other waiting task ,NULL→s2)
t55	(s1—SE, event=0, 0→s1)
t56,t57	(S—SE, event≠0 ,0→s1), S={E,W}
t61	(s3—all x∈I , all event calls return, NULL→s0)

EmGen: An Automatic Test-Program Generation Tool for Embedded IP Cores

Haihua Shen, Yunji Chen, and Jing Huang

Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China
{shenhh, cyj, huangjing}@ict.ac.cn

Abstract: Core-based system-on-chip (SoC) design is quickly becoming a new paradigm in electronic system design due to the reusability of IP cores. However, the validation of IP cores is the most time consuming task in the design flow. This paper presents EmGen, an automatic test-program generation tool designed for embedded microprocessor cores. EmGen provides an configurable formal specification model with heuristic knowledge, which can generate test programs according to different configuration of microprocessors' architecture, a test generation scheme based on heuristic algorithms, which can efficiently provide instructions in test programs, and validation testbenches, which support simulation with generated test programs automatically and check the equivalence of microprocessors and the specified instruction reference model. EmGen is currently in use at ICT for the verification of embedded microprocessor cores. Experiments results show that EmGen can improve verification process and cut down skilled manpower obviously.

1 Introduction

Today, microprocessor vendors are suddenly clamoring to take the lead in offering reusable IP cores ranged from North Bridge to 32-bit microprocessor cores — a trend that promises to shorten the time to market and fill the design productivity gap due to development of deep sub-micron technologies. Compared with general multimillion-gate processors, embedded microprocessor cores pay more attention to low-cost and other attributes such as flexibility and reconfigurability. Traditional design verification comprises a large portion of the effort in designing a processor [1]. Simulation-based verification tries to uncover errors of design by detecting circuits' faulty behavior when deterministic or pseudo-random simulation vectors are applied. Many practices involve in technologies to improve test generation and coverage analysis [2][3][4]. Some papers are more concerned with comparison among those advanced techniques within existing strategies of test generation. It indicates that using random test bench to perform the directed testing is strongly recommended [5]. The primary motivation of our work is to take advanced test generation technologies into practice to adapt to the feature of embedded microprocessor cores' verification. Although it presents a set of new techniques to realize recommendation trends, the main contribu-

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 528-535, 2005.

tion of our work is not in theory, but in describing how to translate this theory into practice in a practical way, a task that is far from trivial.

This paper presents an automatic test generation tool named EmGen, which is able to induce test programs according to design configuration for simplifying the process of verification. EmGen is based on a previously developed configurable formal specification model with heuristic knowledge that can specify request from highly directed tests to completely random ones according to the requirement of different microprocessor cores' configuration and coverage metrics, a reference instruction set simulator, which can provide accurate reference results for DUT verification in executable framework, a test generation scheme with heuristic algorithms, which can efficiently provide appropriate instructions based on the configurable formal specification model, and validation testbenches, which support simulation with generated test programs automatically and check the equivalence of microprocessor cores and their reference model [6]. EmGen's uniqueness, when compared to general-purpose verification tools in stimuli generation [7] [8], lies in the fact that both the generation schemes and the validation testbenches it provides are oriented at the characteristics of flexible and reconfigurable embedded microprocessors by high-level test program generation. It has been taken into practice in the Institute of Computing Technology of the Chinese Academy of Sciences for the verification of a 32-bit embedded microprocessor core.

The remainder of this paper is as follows. In Section 2 & 3, we briefly describe the configurable formal specification model and the heuristic test generation algorithms used in EmGen. Then we present the validation environments and the framework of EmGen in Section 4. Finally we present experimental results in Section 5 and give some conclusions in Sections 6.

2 Configurable Formal Specification Model

Given an RT-level description of a microprocessor core, simulation-based verification requires test programs and a simulator to simulate its execution. The goal we design EmGen is to generate test programs automatically and effectively according to the specified generation rules that describe the requirement of the configurable microarchitecture of embedded microprocessors and the verification plan precisely.

Test generation rules are hard to specify correctly, and yet are often critical and beneficial that their specifications are correct, complete and unambiguous. Formal specifications, based on using multiple constraints to collectively define the behavior of test generation, promise to meet all requirements. Our formal specification is summarized to four style rules.

The "adjustability" rule requires the instructions to be generated fully adapting to different microarchitecture configuration of embedded microprocessor core.

Configurable microprocessor core is an emerging technology that takes the high performance of many different asics or application specific standard products (assp) into an application tailored embedded microprocessor core [9]. General options of embedded microprocessor core configuration can be the size and organization of cache & TLB, the usage of floating point units and multimedia units, the optional bus

interfaces of microprocessor core, the pipeline partition according to the requirement of frequency and so on. According to the adjustability rule, the specification should initialize all parameters in relation to the configuration of microprocessor, as well as parameters of test program length and random seeds.

Constraints on microarchitecture configuration of embedded core are written in the following forms:

```
. Config parameters
   TLB          1
   FPU          0
   MMX          1
   ICACHE       2
   DCACHE       2
. End of config
```

```
.Pipeline stages
   ALU_WB_STAGE     1
   TLB_STAGE        1
   MUL_WB_STAGE     1
   RENAME_STAGE     0
   FMUL_STAGE       3
. End of pipeline stages
```

The "random" rule requires the instructions with or without arrangements to be generated randomly with configurable ratio. Based on the random rule, instruction specifications should be written with appropriate ratio.

The "flexibility" rule requires the instructions to be generated in configurable styles from pseudorandom groups with different seeds to totally directed sequence with strict constraints.

Separated by operands, allowed instructions are divided into four groups. I type includes instructions with only two src operands and dst operand in registers. M type includes instructions that access memory resources. J type & B type are all instructions whose target is to change program counters. The difference between them is that instructions in B type have branch conditions and those in J type have not. Function I/M/J/B (instruction var) is defined to find if the instruction var is in I/M/J/B type. The constraints should be written according to the following reasoning:

$prec(I \cap \tilde{}M \cap \tilde{}J \cap \tilde{}B) \rightarrow R_{src1} \vee R_{src2} \vee R_{dst}$

$prec(\tilde{} I \cap M \cap \tilde{} J \cap \tilde{} B) \rightarrow (R_{base} \vee ((R_{src2} \vee R_{dst}) \wedge \tilde{} (R_{src2} \wedge R_{dst}))) \vee (M_{str} \vee M_{end})$

$prec(\tilde{} I \cap \tilde{} M \cap J \cap \tilde{} B) \rightarrow T_{target}$

$prec(\tilde{} I \cap \tilde{} M \cap \tilde{} J \cap B) \rightarrow R_{src1} \vee R_{src2} \vee T_{offset}$

The prec construct allows the operands of an instruction with expressed precondition to be limited by the expression of constraining logic.

The "divisibility" rule requires each constraint to constrain only the behavior of one instruction component. Equivalently, because the constraining part is isolated from each other, the rule requires the consequent to contain only one or a scope of values for one field.

The four styles above are powerful enough to specify the requirement of generation from random to totally direction, see [6] for more detailed information on the model used here.

Although abiding by the style rules may seem restrictive, it promises many benefits. For example, the specification is easier to maintain. Constraints can be added or removed and independently modified. It is also believed that it is easier to write and debug. Since most existing languages are already written as a list of rules, the translation to this type of specification requires less effort and results in fewer opportunities for errors.

3 Test Generation Process

For a given specification model, EmGen's generation process can be divided into two layers: traversing the internal specification statement tree and generating the instructions on the basis of specification. It is easy to do the statement tree travel in a hierarchical manner. But generating the instructions is more difficult because the generation process interleaved with the execution of Reference Instruction Set Simulator must maintain the intermediate processor states dynamically. The intermediate processor states embedded in all kinds of general registers, control registers, statement registers and memories are always limited by realistic resources in practice. There are several factors in contradiction with each other that may bring on difficulties in generation process, such as the limitation of realistic memory addresses with the supposed large size of idealistic virtual memory addresses. To solve these problems, we present two algorithms named the dynamic boundary check (DBC) algorithm and the scan TLB (ST) algorithm. The DBC algorithm handles the random branches targets to achieve expected quantities and quality of generation according to verification plan. Then the ST algorithm is used to generate enough efficient instructions under the limitation of memory address space in validation environments.

3.1 The Dynamic Boundary Check (DBC) Algorithm

The main idea of DBC algorithm is to change the direction of instruction flows whenever they meet a boundary, carefully select the branch targets to avoid the boundary and delete the dead branch whose target is breaking the rules or terminate the generation process if it does not belong to the random freely group. The DBC algorithm is implemented by following steps.

Step 1: Generate a new instruction according to the constraints of configurable specification. Judge if the instruction is a branch or jump. If the instruction is a branch, then go to Step2. Else go to Step3.

Step2: Search if there is an appropriate target *pc* not in *pc* stack following all the constraints in specification and not breaking the boundary in *pc* stack. If the correct target *pc* cannot be found, then turn to Step4, otherwise, Step5.

Step3: Check the boundary in *pc* stack. If the instruction meets the boundary in pc stack, go to Step6; else transfer the instruction to RISS and go back to Step1.

Step4: Check if the instructions belongs to random freely group. If it is random freely, then cancel this instruction and turn back to Step1 to generate a new instruction instead; else print alert and terminate the generation process.

Step5: Select an appropriate target *pc* for the instruction and transfer the instruction to RISS. Then go on the generation process in Step1.

Step6: If the instruction is in a sequence, terminate the generation process and pop up an alert. Else insert a directly jump instruction behind the instruction to avoid the boundary and go back to Step1.

A case of instruction flows changed by the DBC algorithm is shown in Fig. 1.

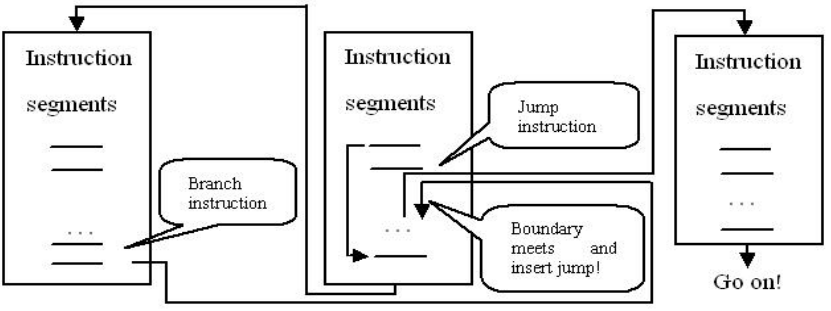


Fig. 1 A case of instruction flows changed by DBC algorithm

The results of experiments show that the DBC algorithm can improve the generation process obviously (see Fig. 3 in section 5).

3.2 The Scan TLB (ST) Algorithm

It is important for EmGen to generate and control all the exceptions and interrupts tightly according to verification plans. In all kind of exceptions, TLB miss happens frequently because of the limitation of TLB pages. To reduce the complexity of testbench, the maxim number and size of TLB pages are always predefined. So it is very important to generate instructions in the limitation of TLB to avoid unexpected TLB miss. The ST algorithm is presented to solve this problem.

Main idea of ST is dynamically initializing the TLB whenever an instruction with a pc in new page or a memory access with a new page address happened. It is necessary to keep a TLB stack and push the TLB page to TLB stack whenever a new TLB page is distributed. When all the TLB entries are distributed, new instructions program counter and memory access is limited to the old blank TLB. The ST algorithm is described as following:

Step1: Initialize a TLB page according to the predefined first pc in the configure file.

Step2: When generating a new instruction, search the pc in TLB stack. If the instruction is on a page distributed, do nothing to TLB stack. Otherwise insert the TLB entry to the stack. When TLB stack filled, go to Step 5.

Step3: Check if the new instruction is a branch or jump. If the instruction generated in Step2 is a branch or jump, then check if the page that contains the branch target is in TLB stack. If in the stack, do nothing. Otherwise check if the TLB stack filled. If filled, regenerate the same instruction with different target by several times (Retry times can be predefined or changed dynamically in progress according to generation plan). If there is an appropriate target whose page entry is already in TLB stack, then do nothing. Else go to step 5. If the TLB stack is not filled, insert the page that contains the instruction target into TLB stack.

Step4: Go back to Step1.

Step5: Terminate the generation process because of the TLB miss.

The ST algorithm acts as supplementary for the DBC algorithm to improve generation process in EmGen. Experiment results show that the required number of instructions can be generated easily according to the configurable specification by means of DBC and ST algorithms (see Figure 3 in Section 5).

4 The Sketch of Validation

EmGen's modeling framework contains several components: an instruction library, a configurable formal specification model with a parser, a test generator, a reference instruction set simulator (RISS), a simulator with the validation environment, an RT-level design under test (DUT), and a result compare logic. The overall architecture is shown in Fig.2. All microprocessor instructions are described in an instruction library. The configurable formal specification model includes the configurable formal specification described above and a parser, which can parse the specification to internal data structures. Test generator selects instructions from library according to the configurable formal specification, which can be optimized by the corresponding coverage metrics. Besides, a validation environment, which supports simulation with generated test programs and checks the equivalence of microprocessor cores and their reference model automatically, is also provided by EmGen. By means of the validation environment, EmGen can judge whether the results of the test runs are correct or not.

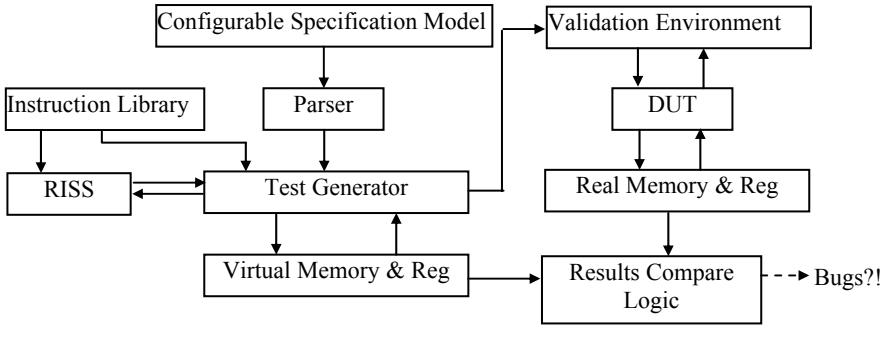


Fig. 2. The sketch of EmGen

5 Experimental Results

To demonstrate EmGen on a meaningful design, we chose a 32-bit embedded RISC microprocessor core based on MIPS instruction set with about 0.48 million gates of logic as DUT for verification. The optional configurations of the microprocessor core are 2-Way Set Associative I-cache & D-cache, 4-Way Set Associative I-cache & D-cache, no Cache & TLB, the floating point units and multimedia units. Simulations

were run on Intel Pentium 4 2.8GHz HyperThreading system with 2G of main memory.

Verification results: Before presented approach is taken into practice, the DUT has been verified for more than two months. Stimuli include popular benchmarks (Whetd, Dhrystone, Paranoia), real operation systems (Linux) and many manual test programs. Using the random test programs provided by the methodology, two previously unreported bugs have been found in the DUT. The process of finding bugs is now nearly automated and much easier since result compare logic can give direct location of errors.

Performance results: Performance issues, such as speed and memory usage, do not pose to be problems because of the hardware improvement. So we are free to focus on generating interesting simulation inputs. It has been mentioned above that effects of generation process is influenced by resources limitation. Predefine an address space limitation available in test program generation to be 64x8Kbytes. Our experiments show that the maximum number of instructions generated by DBC&ST algorithms is an order of magnitude larger than it was without DBC&ST algorithms (see Fig. 3).

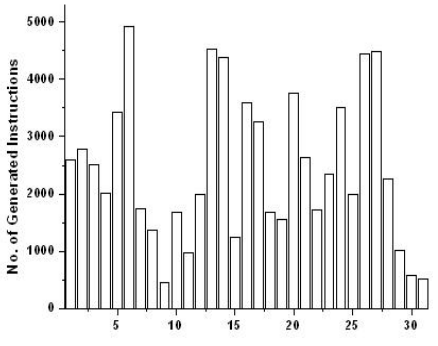
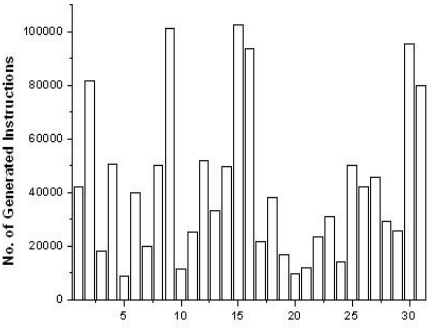


Fig. 3 a) Ability of random test generator without DBC&ST Algorithm



b) Ability of random test generator with DBC&ST Algorithm

Coverage results After running more than 55,000,000 instructions generated, the average code coverage of the design reaches 73.8% of line coverage, 68.9% of condition coverage, 59.78% toggle coverage, which is hard to be improved by increasing more random instructions. It is obviously that the likelihood of generating more effective events so unintentionally is low, though the EmGen is capable of generating such events. To improve the coverage of simulation, the design should be simulated with more expert experiences, and so configuring such constrained corner cases supported by configurable formal specification model to determine biases is extremely fruitful; most of the "holes" can be covered with these biasing.

6 Conclusions

In this paper, we describe EmGen, an automatic test-program generation tool designed for configurable embedded microprocessor cores. EmGen's modeling platform includes an configurable formal specification, which can describe the requirement of microprocessor configuration and verification plan completely and correctly, a heuristic test program generator with DBC and ST algorithms, which enable the generation of test programs across the full spectrum, from completely random through testing-knowledge biased random, to highly directed, a reference instruction set simulator, which can provide reference results for all test cases, and a validation environment, which check the equivalence of simulations and reports the bugs automatically. EmGen has been taken into practice for the verification of a 32-bit configurable embedded microprocessor core. Experiment results have proven its flexibility, applicability and good performance.

References

1. D. Campenhout, T. Mudge, J. Hayes, "High-Level Test Generation for Design Verification of Pipelined Microprocessors", In proceeding of the 36th ACM/IEEE Design Automation Conference (DAC), (1999) 184-188
2. S.Fine, A. Ziv, "Coverage Directed Test Generation for Functional Verification Using Bayesian Networks", In proceeding of the 40th ACM/IEEE Design Automation Conference (DAC), (2003) 286–291
3. R. Emek, et al. "X-Gen: A Random Test-Case Generator for Systems and Socs", IEEE International High Level Design Validation and Test Workshop, Cannes(2002)
4. O. Lachish, E. Marcus, et al. " Hole Analysis for Functional Coverage Data", In proceeding of the 39th ACM/IEEE Design Automation Conference (DAC), (2002) 807-812
5. M. Bartley, D.Galpin, T.Blackmore. " A Comparison of Three Verification Techniques: Directed Testing, Pseudo-Random Testing and Property Checking", In proceedings of the 39th ACM/IEEE Design Automation Conference (DAC), (2002) 819-823
6. Haihua Shen, et al. "Adaptive Test Program Generation for Embedded Microprocessor Core", In proceedings of the 1st International Conference on Embedded Software and System (ICESS), (2004) 472-479
7. Synopsys, Inc. "Constrained-Random Test Generation and Functional Coverage with Vera", http://www.synopsys.com/products/vera/vera60_wp.pdf, (2003)
8. Verisity Design, Inc. "Specman Elite", http://www.verisity.com/products/specman.html, (2004)
9. Jim Lipman, "Configurable SoCs Give You Options", http://www.techonline.com/commnity/related_content/11384, (2000)

Formal Verification of a Ubiquitous Hardware Component

Lu Yan

Turku Centre for Computer Science (TUCS) and
Department of Computer Science, Åbo Akademi University,
FIN-20520 Turku, Finland
Lu.Yan@abo.fi

Abstract. This paper is a case study of the verification of a seven-segment LED display decoder circuit design, in which two popular verification tools, HOL and PVS, are compared and evaluated.

1 What Is Formal Hardware Verification

We consider a formal hardware verification problem to consist of *formally establishing that an implementation satisfies a specification.* [1] The term *implementation (Imp)* refers to the hardware design that is to be verified. This entity can correspond to a design description at any level of the hardware abstraction hierarchy, not just the final physical layout (as is traditionally regarded in some areas). The term *specification (Spec)* refers to the property with respect to which correctness is to be determined. It can be expressed in a variety of ways - as a behavioral description, as an abstracted structural description, as a timing requirement etc. [2]

2 The Ubiquitous Hardware Component

We illustrate our experiences with formal verification in ubiquitous hardware design via a comparative case study of the verification of the circuit design of a ubiquitous hardware component: seven-segment LED display decoder.

A seven-segment LED display is comprised of seven light emitting diodes (LED). Input signals are applied to the input port of the seven-segment decoder, and the decoder translates them into ON/OFF status of the seven LEDs. Then, selected combinations of the LEDs are illuminated to display numeric digits and other symbols as shown in Figure 1.

The primary function of the decoder is to turn on/off corresponding LEDs based on inputs. Let W, X, Y, Z represent the input port of the decoder, then we get sixteen possible combinations of the four input signals, which means any digit (0 - 9) and some letters (A - F) can be displayed on the seven-segment LED display. Let a, b, c, d, e, f, g represent the output port of the decoder, and let on be 1 and off be 0, then we can create a truth table like Table 1 for describing the intended behavior of the decoder.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 536–541, 2005.

Table 1. Truth table for the switching function

Display	Input ($W\,X\,Y\,Z$)	Output ($a\,b\,c\,d\,e\,f\,g$)
0	0000	1111110
1	0001	0110000
2	0010	1101101
3	0011	1111001
4	0100	0110011
5	0101	1011011
6	0110	1011111
7	0111	1110000
8	1000	1111111
9	1001	1111011
A	1010	1110111
B	1011	0011111
C	1100	1001110
D	1101	0111101
E	1110	1001111
F	1111	1000111

Intuitively, the abstraction of seven-segment decoder is a four-input seven-output switching function. One possible approach is to build up the circuit directly from the specification, but here we consider another approach based on partition-and-merge algorithm.

First we divide the four-input seven-output switching function into seven four-input one-output normal functions, then implement each function separately. When all functions are ready, we put together all parts and get the final implementation. In this way, the complexity of the design task is greatly reduced. The drawback is probably some redundancy, but this can be refined in the final merging stage.

We shall go into more details of the implementation of one part as an example. The representation function is the abstraction of the intended behavior of LED a, which takes four input signals W, X, Y, Z and generate one output signal a correspondingly, as shown in Table 2.

With the initial implementation, we can refine it with a Karnaugh map. The process is illustrated in Figure 2. Although this refinement result is good enough, we should also consider more practical issues like technology, cost, etc. Here we choose to make the design mainly with NAND gates:

$$a = \overline{\overline{Y\,\overline{Z}} \cdot \overline{X\,\overline{Z}} \cdot \overline{\overline{W}\,Y} \cdot \overline{X\,Y} \cdot \overline{W\,\overline{Z}} \cdot \overline{\overline{W}\,X\,Z} \cdot \overline{W\,\overline{X}\,\overline{Y}}}$$

Then it is time to translate the refinement result into schematic design. The diagram is straight forward, as shown in Figure 3. The final step is to design the real circuit based on the schematic design. Here we choose the NAND gate model and the proper CMOS tool as the atomic element to build up the whole

Table 2. Truth table of 4-input function

Input ($WXYZ$)	Output (a)
0000	1
0001	0
0010	1
0011	1
0100	0
0101	1
0110	1
0111	1
1000	1
1001	1
1010	1
1011	0
1100	1
1101	0
1110	1
1111	1

design. The result is shown in Figure 4. Now the design task of the first part is completed. With the same method, we can design the other six parts.

3 A Comparison of HOL and PVS

Generally, although HOL and PVS are similar to each other and shares a lot of common features, partly because they are all based on higher order logic and for supporting formal methods applications with proof, there are still some differences. In this section we wish to discuss in some detail our own, more personal, experiences with regards to the case study:

- The meta-language of HOL is ML; hence HOL type system is similar to the type system of ML, which form the basis of the higher order logic theory. PVS is written in Lisp and implements classical typed higher order logic with an extension of predicate subtypes. PVS has many built-in types and uses type constructors to build complex types.
- The specification language of HOL is a ML-style one, which uses the ML datatype `term` to represent the HOL logic; theories are created in ML functions by `new_definition`.

```
val NOT_DEF =
  new_definition("NOT_DEF",
    (--‘NOT a x = (x = ~a)‘--));
```

  Take a look into the case study, we can see that the specification consists of the hardware components specification, the target hardware device specification composed with above components' specification, (and the correctness relationship to be proved by `set_goal`, which looks like a part of the proof).

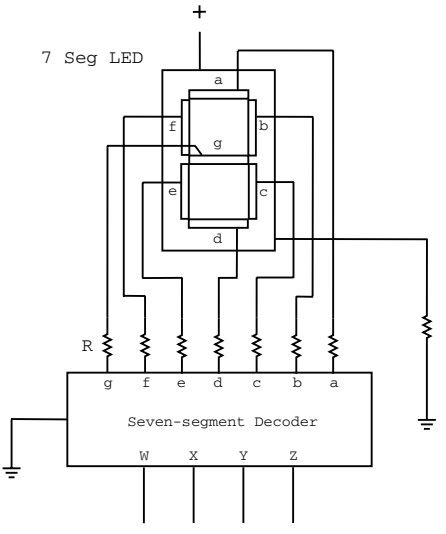


Fig. 1. Hardware Component

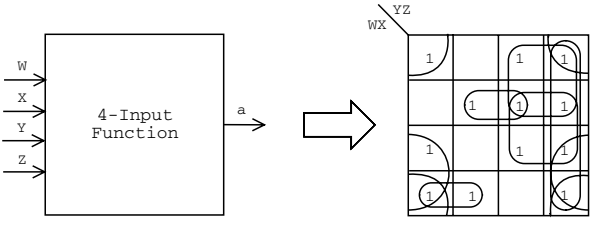


Fig. 2. Karnaugh map for 4-input function

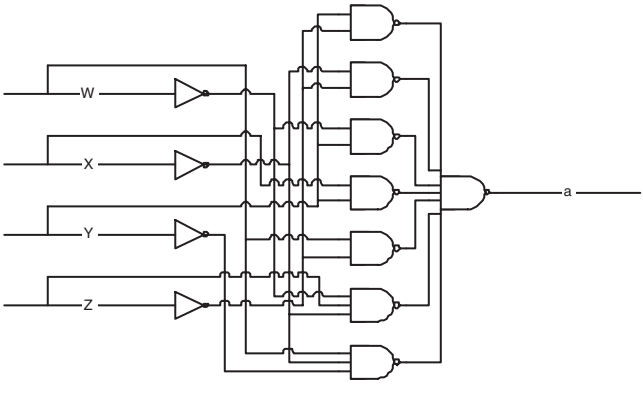


Fig. 3. Schematic Design

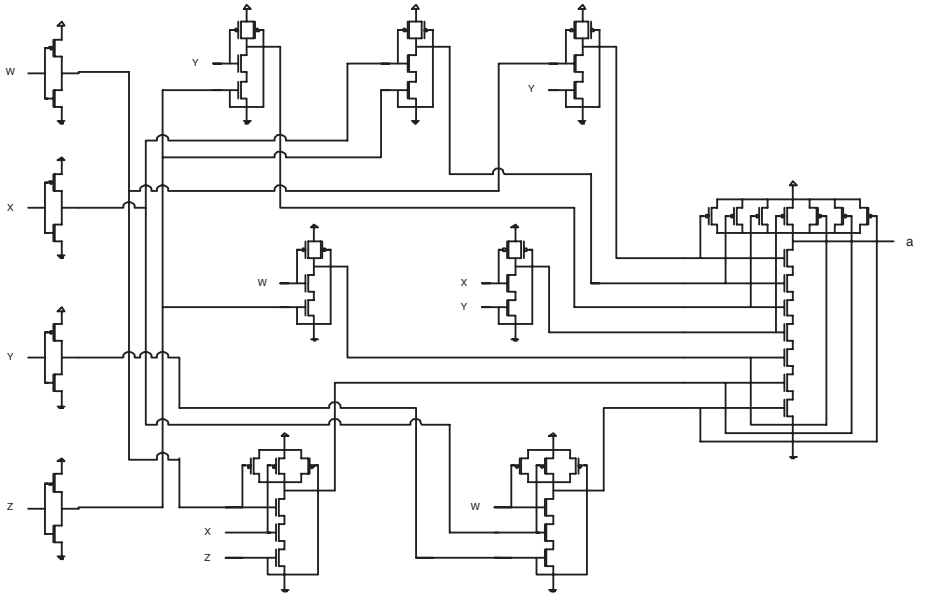


Fig. 4. Circuit Design

```
set_goal([],(--'!w x y z a.
  LED_A_IMP w x y z a ==> LED_A_DEF w x y z a'--));
```

The specification language of PVS is rich, containing many different type constructors and predicate subtypes. Unlike HOL, the syntax is more fixed; many language constructs, such as `IF` and `CASES` are built-in to the language. A specification is usually divided in several theories and theories can import other theories. Although from the case study, we can find out that the specification is organized similarly, there are two obvious differences:

- Variables have to be declared before using (there is no default datatype mechanism for undefined variables).

  ```
  % input and output
  W, X, Y, Z, a: VAR bool
  ```

- The correctness relationship to be proved is within `THEORY`.

  ```
  logic_gates: THEORY
  ⋮
    implementation_correctness: THEOREM
      imp(W, X, Y, Z, a) IMPLIES spec(W, X, Y, Z, a)

    END logic_gates
  ```

– HOL supports both forward and backward proving, but it emphasizes on backward proving by supplying many useful tactics for it. A tactic transforms the proof goal into several subgoals. HOL has a large collection of tactics as well as many proving tools. In the process of proving, we need to load such tools from libraries by `load` before proving because they don't automatically "stand forward" when applicable.

```
load "bossLib";
load "simpLib";
load "mesonLib";
```

A thorough look of HOL libraries beforehand will help us to get familiar with some of powerful proving tools.
PVS has many tools in the core system which can be automatically invoked. We are quite impressed in the process of proving; such tools are built-in to the system and are ready to use by invoking `grind` etc.

```
implementation_correctness :

  |-------
{1}   FORALL (W, X, Y, Z, a: bool):
        imp(W, X, Y, Z, a) IMPLIES spec(W, X, Y, Z, a)

Rule? (grind)
```

Another difference is that after supplying a tactic, the system repeatedly apply it to the current goal until no changes in the current state. A PVS tactic is like a `REPEAT` HOL tactic in this way.

```
e(REPEAT GEN_TAC);
```

– The most famous difference between HOL and PVS is that the former is a LCF-style prover, which has better security, user extensibility and also ways to import and export proofs to other provers.

Acknowledgement

The author is deeply grateful to Joakim von Wright for his kindly support and assistance.

References

1. C. Kern and M. Greenstreet. *Formal Verification in Hardware Design: A Survey.* ACM Transactions on Design Automation of Electronic Systems, Vol. 4, April 1999, pp. 123-193.
2. A. Gupta. *Formal Hardware Verification Methods: A Survey.* Formal Methods in System Design, Vol. 1, pp. 151-238, 1992.

Model Optimization Techniques in a Verification Platform for Classified Properties*

Ming Zhu, Jinian Bian, and Weimin Wu

Department of Computer and Science, Tsinghua University, Beijing, 100084, China
zhum99@mails.tsinghua.edu.cn
{bianjn, wuwn}@tsinghua.edu.cn

Abstract. In system functional verification, collaborative method is an effective technique. Co-SAM verification platform employs three verification methods, static analysis, logic simulation and model checking, to implement verification process. With the collaborative platform, system properties are classified and verified. A whole model for a system inclines to ineffectiveness and states explosion during verification. Model refinement is indispensable. This paper presents special optimization techniques for verification model refinement, i.e. property grouping, signals reordering, and model hierarchizing. Experimental results demonstrate the validity of these optimization techniques in system functional verification.

1 Introduction

With the incessant increase of IC design, system functions verification become the main bottleneck in most design flows and consumes up to 70% of the time required for a system design [1]. Usually, a project needs to be staffed with three verification engineers for a design engineer [2]. The overall idea of verification is to find as many bugs as possible as in earlier design stage and get to an experience-based confidence level. However, large scale and complexity of systems hinder the possibility.

Current design verification strategies span a wide spectrum that ranges from brute-force manner, such as simulation, or random test generation techniques to formal verification methods, such as equivalence checking and model checking [3].

Traditional event-driven simulation, mastered by verification engineers, is effective for medium-scale systems, and suited for verifying units. As gates count increases to several millions, the number of simulation cycles explodes by orders of magnitude. The gap between simulation cycles and increasing gates count is shown in Fig.1 [2]. Besides complexity, another disadvantage of simulation-based verification is that test benches only cover a limited subset of the entire states. Even if expected functions are verified by artificial test benches and unexpected bugs by random generated ones, some fatal bugs may be found later for incompleteness of simulation. An

* This research was supported by the National Natural Science Foundation of China 60236020, 60273011 and Hi-Tech Research & Development (863) Program of China 2003AA115110.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 542-548, 2005.

attractive alternative to simulation in verification is formal approach [3] which statically verifies a design without using test benches. *Model checking* [3] is one of important formal methods for automatically verifying finite state systems. Compared with simulation, the superiority of model checking is full automatic and with useful counterexample as by-product. On the other hand, the problem of model checking is *states explosion* that constantly occurs when a system has many wide-data.

Many improved strategies have been proposed in simulation or formal verification for covering the shortage and employing the advantages of different approaches. Developed by Ganai et al.[4], *SIVA* employs the combination of ATPG and BDDs to generate input vectors, and significant improvement in state space coverage is reached. An *extended FSM* model for RTL designs proposed by Huang et al. [5] can be automatically extracted from codes and formally analyzed for the vector generation.

Neither simulation nor model checking can cope with very large and complex system alone because of inherence disadvantage. The natural idea is to combine the two techniques into a unified framework. Many hybrid methods are proposed for various applications. Hazelhurst et al. [6] proposes a hybrid technology combining many formal methods to reduce the cost of verifying designs with complex initialization. Presented by Ho et al. [7], *Ketchum* integrates symbolic simulation with SAT-based BMC and provides the ability for 4500 latches and 170K gates.

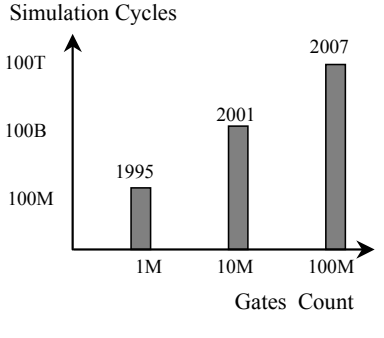


Fig. 1. Simulation cycles vs. gates count

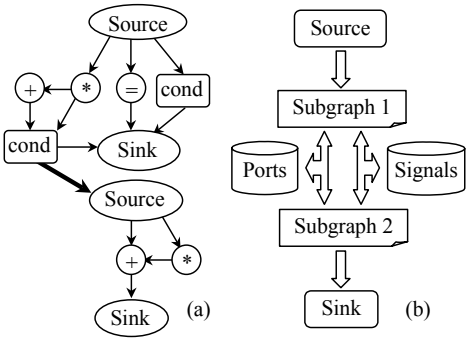


Fig. 2. CDFG structure and data flow

Co-SAM [8], our collaborative verification platform combines simulation and BDD-based model checking on a model from CDFG. Special model optimizations are adopted. The remainder of this paper proceeds as follows: Section 2 depicts the verification platform; section 3 discusses the optimization techniques for the platform; properties verification results and conclusion are presented in Section 4 and 5.

2 Co-SAM Verification Platform

The emphasis of existing work focuses on verification model creation and searching algorithms. Our research aims to *classified property verification* and *special model refinements* on CDFG (**C**ontrol **D**ata **F**low **G**raph) structure.

2.1 CDFG Structure

In our verification platform, Co-SAM (**Co**-verification of **S**imulation, **A**nalysis, **M**odel checking), CDFG structure bridges the gap between simulation and formal verification. CDFG depicts the behavior of a target system naturally and intuitively through CFG and DFG subgraphs. CFG represents conditional branching, iterations, and modules, and DFG describes operations and data dependencies. Each subgraph contains a source and a sink node (as Fig.2 a shown). Vertices and edges in subgraph record the dependency of data and control respectively. All referenced ports and signals are attached to the subgraph (see Fig.2 b); hence it is easy to recognize whether a subgraph is influenced by the signals in a property. The division of CFG and DFG provides a remarkable advantage for data operations in simulation and FSMs creation in model checking. Furthermore, the system hierarchy can also be remained in CDFG. A critical model for property verification in Co-SAM platform is refined from CDFG.

2.2 Property Classifying Principles

An *assertion* is a specification, which states what a design should be satisfied with or not. An execution of a system is formalized as an infinite sequence of states, and any set of such sequences is a *property*. Property checking checks a system to ensure its properties satisfiability. Properties are usually expressed with logical formalism, such as *CTL* language and *OVL* assertions. For multiple engines co-verification, properties should express not only data operations for simulation but also temporal logic for formal method.

Simulation can deal with data operations effectively, especially in spite of data width that is fatal for model checking. Model checking is attractive and promising in verifying FSMs. The exhaustive exploration in all possible states space overcomes the well-known code and event coverage limitations of simulation. But it is suitable only for medium-scale systems with specified interfaces clearly.

Inheriting OVL and CTL, improved descriptions of properties are defined in Co-SAM system; leveraging static analysis, simulation and model checking, properties are classified to employ their good qualities; these properties are divided into three classes as following rules: analysis properties deal with the condition translation, simulation properties manage data processing, and model checking properties dispose temporal logic. System properties can be embedded into processes or parallelized with them. Although properties are classified, it does not mean that one property can only be verified with a certain method. The supplement among different verification engines is necessary so that the bugs can be found as many as possible.

2.3 Collaborative Verification Constitution

Some conclusions can be drawn according to above analysis: simulation method is good at modules and data operations, time-consuming in large designs and test-benches generation; model checking is highly automatic and unique in temporal logic,

but consumptive hugely in states space, especially for wide-data operations. An effective verification platform should employ their good qualities.

Our collaborative verification scheme is shown as Fig.3. First the properties are classified automatically for static analysis, module simulation and model checking. Then system properties are preprocessed and CDFG model is generated which provides more effective operations and facilitated structure. Next, several refinement techniques on the model are adopted. At last, static analysis, simulation, and model checking are performed on the refined model.

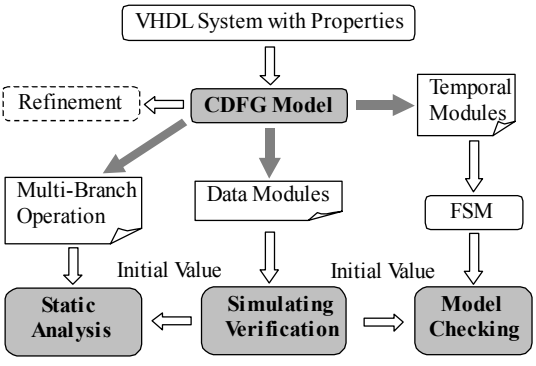


Fig. 3. Functional verification flow

With the help of hierarchical structure of CDFG, it is easy to trace a multi-branch signal and partition the whole design into modules, to find the value conflicts, and to check the branch conditions. Simulation process can provide some initial values for model checking and static analysis. More detailed information about the collaborative platform is introduced in [8].

3 Optimization Techniques in Model Refinement

The CDFG from system directly involves some redundant information that will increase the size of model, so refining and optimizing operations are indispensable.

3.1 Properties Grouping and Signals Reordering

For a large design, it is almost impossible to create a whole model; therefore a refined sub-model is competitive and practical. In general, a system module is not relative to all inputs and outputs. Simultaneously, each input signal only impacts a cone-shaped region, and each output signal is controlled by partial inputs and state signals, as shown in Fig.4. Usually a property's checking will not act on the whole system, and the referred signals are limited, named *property's localization.* For instance, in an 8-bit counter, high 4-bit value can be ignored when a property only involves low 4-bit.

Prior to verification work and according to property's localization, grouping the properties that refer the closed set of signals into several and creating independent models is a practical way to diminish model size. The signals that do not exist in current group of properties and are not referred in corresponding modules can be removed temporarily from the model. A sub-model has fewer states for verification than the original one. The result in Table 1 shows the comparison between original internal results and ones after model refinement, properties grouping.

Table 1. Effect of properties grouping

Circuits Name	Sigs Num	Nodes count			Bytes count		
		Original	Group1	Group2	Original	Group1	Group2
B01	10	340	330	325	74848	73664	73360
Count16	33	196	73	65	73320	68808	68296
Ab128	386	26188	25296	25479	594152	504056	533048
Ab160	482	42708	38788	38741	984104	732024	726136

In Table 1, the properties are grouped into two sub-groups according to their characters. The first two columns are the test benches name and signals number in a design. Other columns list the internal nodes and bytes cost by the corresponding model. After the grouping operation, the signals that are not referred to will be removed temporarily, and the internal nodes and bytes count during verification are both reduced for *one* group. Although the sum of *two* groups exceeds the original one, this still provides a practice way to deal with larger systems.

The properties grouping effect depends on the signals relation among properties. For *Count16,* two groups are separated well and the nodes count is decreased obviously, whereas properties in *B01* are much closed and the count changes little.

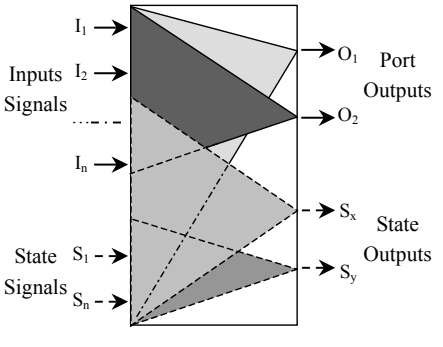


Fig. 4. Variable local characteristic

```
Algorithm module_signals (property) {
  sigs_list L ← ∅ ;   marked_sigs M ← ∅ ;
  L ← get_sigs(property);
  for each v ∈ L do {
    if v ∉ M then {
      sigs_list V ← transitionF(v) ∪ outputF(v);
      for each w ∈ V do {
        if w ∉ L  then L ← L ∪ {w};
        M ← M ∪ {w};
      } } }
  return L; }
```

Fig. 5. Module signals extracting algorithm

The initial signals order is critical for BDD scale in model checking. Although many algorithms provide the ability to reorder variables, a good initial order will save the cost of reordering. Before model checking, the signal orders arranged with heuristic information, not with their defined sites, provide a relatively satisfied one for BDD creation in model checking.

After signals reordering, the internal model and time cost are both diminished as Table 2 shown. The variables are reordered according to their appearance and action, and they are closed to the aim orders for BDD generation. Thus the process of signals recording in model checking costs less time than before as the *Ratio* column shows.

Table 2. Effect of signals reordering

Circuits	Sigs	Nodes count		Bytes count		Reorder time cost		
Name	Num	Before	After	Before	After	Before	After	Ratio
B01	10	340	286	74848	71344	0.00	0.00s	/
Count16	33	196	125	73320	47940	0.16s	0.04s	25%
Ab128	386	26188	10541	594152	267832	31.87s	6.69s	21%

Fig.5 shows the algorithm that finds out the indispensable signals in a sub-model for properties grouping and signals reordering.

3.2 Model Hierarchizing and Refining

In model checking, each module is formalized as a FSM, and all the modules in a system compose tremendous product-FSMs. A hierarchical simplification for model refinement is adopted in our platform, and the contents which are not referred to by current properties group will be encapsulated and hidden until they are called by subsequent properties. Then the system FSMs is remarkably decreased. To accelerate the updating process for FSMs, variables-substituting refinements are performed, and to reduce the amount of search effort, states space pruning techniques are also adopted.

The blocks that have close relations will be encapsulated into a module. In table 3, last three columns show the effect of model hierarchizing. *Nodes count* is reduced markedly: internal nodes count is reduced to about 70% than before. It is also a valuable assistant to solve large-scale systems. These optimization techniques are employed corporately in the verification platform.

Table 3. Effect of model hierarchizing

Circuits	Sigs	Potential	Reachable	Nodes count		
Name	Num	states	States	Before	After	Ratio
B01	10	1024	432	340	257	76%
Count16	33	4.295e+09	4.295e+09	196	131	67%
Ab128	386	3.940e+115	2.463e+114	26188	18347	70%

4 Properties Verification Results

The functional verification scheme is performed on *Traffic Light Controller* (TLC) which is designed for traffic light controller in an intersection between a highway and

a farm road, and includes *controller* and *timer* processes. The properties verification results are listed in Table 4.

The verification result *True* means the design is satisfied with the property, and *False* means the contrary. The verification type means the engine which is selected to verify the property. Three methods can not only work separately, but also overlap each other. Such as the properties, No.2 for TLC, can be verified with simulation and static analysis respectively. It enhances the ability to find as many bugs as possible.

Table 4. TLC benchmark verification

Properties Description	Verification Type	Result
assert_range(number, 0, 10)	Simulation	True
assert_ifcond(number>10, T_out_short= true)	Simulation	False
assert_ifcond(number>10, T_out_short= true)	Static Analysis	False
assert_EG(counter= 8 and T_out_long= true)	Model Checking	False
assert_AGpAF(car_farmroad, farmroad=green)	Model Checking	True

5 Conclusion

In embedded system verification, simulation and emulation are used widely, but the ability is not quite enough. Promising formal verification is energetic, whereas the shortcoming of space explosion restricts its application widely.

In this paper, we propose a novel paradigm that combines good qualities of simulation in data operation and model checking in temporal logic. In the new scheme, with CDFG as the footstone, properties are classified and verified with corresponding verification engines. Some optimization techniques are adopted for model refining, and the verification scale and efficiency are both improved. The experimental results demonstrate their effectiveness.

References

1. TransEDA, Foundation Models - System Level Verification IP. http://www.transeda.com
2. Saeed Coates. Assertive Verification: A Ten-Minute Primer. http://www.EEDesign.com
3. E.M. Clarke, J.O.Grumberg, and D.A. Peled. Model checking. MIT Press, Mass (1999)
4. Malay Ganai, Adnan Aziz, and A. Kuehlman. Enhancing Simulation with BDDs and ATPG. In: Proceedings of 36th DAC, New Orleans, LA, June (1999), pp. 385-390.
5. R.C-Y Huang, K-T.Cheng. A New Extended Finite State Machine (EFSM) Model for RTL Design Verification. In Proc. of IEEE IHLDV and Test Workshop (1998)
6. S.Hazelhurst, O.Weissberg, G.Kamhi, etc. A Hybrid Verification Approach: Getting Deep into the Design. In Proc. of 39th DAC (2002) 111-166
7. Pei-Hsin Ho, T.R.Shiple, K.Harer, etc. Smart Simulation Using Collaborative Formal and Simulation Engines. In Proc. of ICCAD (2000) 120-126
8. Ming Zhu, Jinian Bian, Weimin Wu. A Novel Collaborative Scheme of Simulation and Model Checking for Property Verification. In Proc. Of CSCWD (2004)

Using Model-Based Test Program Generator for Simulation Validation

Youhui Zhang, Dongsheng Wang, Jinglei Wang, and Weimin Zheng

Dept. of Computer Science, Tsinghua Univ.
100084 Beijing, P.R.China
zyh02@tsinghua.edu.cn

Abstract: The continuous advances in microelectronics design are creating a significant challenge to design validation in general. Tackling pipelined microprocessors is remarkably more demanding. This paper presents a methodology to automatically produce a test program for simulation-based validation of microprocessors maximizing the given verification constraints. The approach integrates an accurate c-simulator to trace internal states, including memory access patterns, cache states, pipeline states and so on, of the target processor to generate test vectors with higher efficiency. The test program generator is integrated into a co-verification environment, which is used to verify an embedded processor with a 7-statge pipeline developed by our team and gained remarkable effects.

1 Introduction

It is widely recognized that functional verification emerges as the bottleneck of the CPU design development cycle. The cost of the late discovery of the recently found Pentium FDIV flaw (around $475,000,000) demonstrates the implications of having a design that does not totally conform to its architectural specification [1]. It is therefore not surprising that, for a typical microprocessor design project, up to half of the overall resources spent, are devoted to its verification [2, 3].

Although many techniques have been proposed in the past (e.g., static checks, formal verification [4] [5], mutation testing [6]), none has gained enough popularity to compete with the current practice of validation by simulation. Designers typically resort to extensive simulation of each design unit, and of the complete system, in order to gain confidence over its correctness.

Many metrics have been proposed to evaluate the thoroughness of a given set of input stimuli, often adopted from the software testing domain [7], ranging from statement or branch coverage, to state coverage (for finite state machine controllers). Many variants have been developed, mainly to cater for observability [8] and for the inherent parallelism of hardware descriptions [9], which are not taken into account by standard metrics.

But the right trade-off between designer's time and validation accuracy is often difficult to find, and this often results in under-verified systems. A top-down system

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 549-556, 2005.

level design methodology implies two requirements on a test-bench generation methodology:

- It is mandatory to validate the system as early as possible;
- Validation vectors should be reusable, i.e., the same test bench should be used during design validation at every level of abstraction.

This paper suggests a methodology for the functional verification of microprocessors, and explains how one model-based method is employed to construct the internal state machines of the target processor to generate test vectors with higher efficiency. In addition, some features are introduced, including instruction trees, support of jump instructions and so on. Based on the methodology, a hardware/software co-verification environment is implemented that the system was validated in the architecture design phase and testing vectors could be reusable. Therefore, it reduces significantly the functional verification period of the THUMP107 CPU, which is a MIPS 4KC like embedded processor with highest frequency of 500MHz.

Different from those methods that evaluate the testing coverage when vectors are executed, our implementation can evaluate vectors when they are being generated. That is, as soon as every vector is generated, it will be executed by an accurate CPU c-simulator and the internal states of the target processor can be traced clearly. Then the generation policy will be tuned to produce the next vector with higher coverage. Because the running speed of c-simulator is much higher than the RTL-level simulation, our implementation owns higher performance as well as satisfactory coverage.

Some additional features are implemented as follows.

1. Owing to a friendly GUI, users can set rich constraint conditions to guide the generation. For example, users can specify the type of instructions to be produced, the address range of memory access or whether the transition of some internal states is tested or not.
2. The test program generator is integrated into a co-verification environment where testing traces generated by the RTL-level simulation can be compared with those from the c-simulator directly to locate bugs if existed.

The remainder of the paper is organized as follows. Section 2 presents the architecture of our test bench generator, especially the c-simulator. Section 3 introduces the generation approach, including the organization of the instruction tree, the FSM of the target CPU and some other mechanisms. The co-verification environment is described briefly in Section 4 and some conclusions are drawn in the last section.

2 Architecture

2.1 System Frame

The architecture of our test program generation is presented in Fig. 1. *GUI* is used to specify the generation constraints, including target instruction types, the address range of memory accesses, whether the transition of some internal states is tested, etc.

Instruction tree is a set of the templates of target instructions. For one certain type, the constraints of its operands are stable, so the specification of different instructions is defined in the tree used to create testing vectors with different valid operands.

C-simulator is the major component of our generator. It is employed to perform the generated instructions to trace the transition of internal states, which are maintained by *FSM* and *Resource* modules.

Some major components of the target CPU, including the memory, register files and caches, are maintained by *Resource* module. It means that their current states and history logs are recorded, which can be used to allocate the proper resource to fulfill the request from *Generator*. The effect of *FSM* is similar, that is, the internal states of memory access patterns, cache states, and pipeline states are maintained.

Generator is the coordinator of the whole system. At first, it accepts users' constraints and accesses *Instruction tree* to generate the first instruction randomly. Then, the instruction is performed by *c-simulator* that modifies the states of *FSM* and *Resource* properly. According to the latest states, *Generator* can repeat the former steps to produce instructions satisfying constraints. For example, if the next instruction is required to be RAW data dependent with its preceding one, *FSM* and *Resource* are checked to find the register written by the preceding instruction and then the *tree* will be visited to generate a proper one to fulfill the requirement.

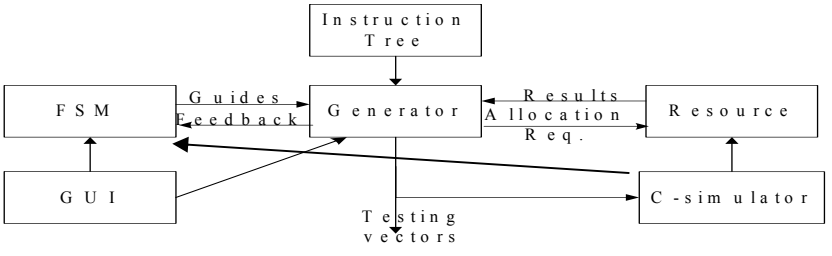


Fig. 1. The framework of the test program generator

2.2 C-Simulator

As ASIC designs explode in size and complexity, the traditional RTL to layout design and verification flow prove inadequate for these multi-million gate systems. We are moving towards extending this flow by concentrating our design and verification efforts before the RTL to layout flow comes into the picture. Therefore, one cycle-accurate processor simulator written in C++ language, THUMPSim, was implemented when our research team began to design a MIPS 4KC like embedded processor.

This simulator is an accurate C-model of our CPU, which implements the micro-architecture of the processor and some basic peripherals. The driven engine of THUMPSim is specially designed for processors, in which an event-driven signal update algorithm is achieved to simulate all hardware activities in every cycle.

THUMPSim contains the following parts.

1. All internal signals
2. All major functional components of the processor, including ALU, MMU, BIU and so on.

3. The register files and cache.
4. All pipeline stages

Because THUMPSim accurately simulates the target processor, the transition of its internal states driven by input instructions is as same as the design target. This feature is used to guide the production of testing vectors to best satisfy the given constraints, which will be described in the next section.

3 Test Bench Generation

3.1 Instruction Trees

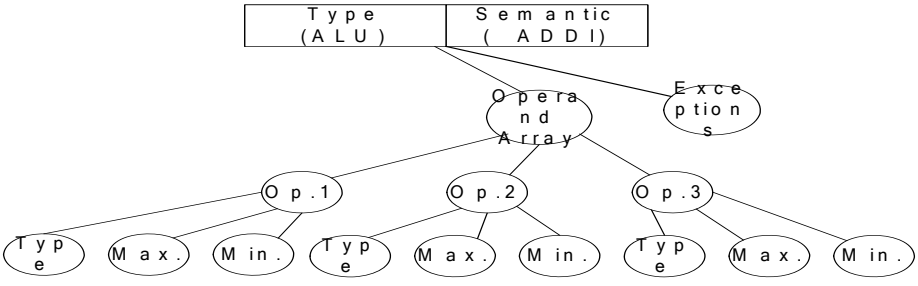


Fig. 2. The tree of *ADDI*

Instructions are modeled as trees at the semantic level of the processor architecture. Generation of instruction instances is done by traversing the instruction tree. The tree of one instruction includes its type and a semantic procedure at the root, its operand array and possible exceptions as internal nodes. Under the array node, the detailed information of all operands is listed. *ADDI* has three operands as described in Fig.2. The constraints of each of these three, including the operand type and the Max./Min. values are presented as leaves. In our definition, there are three different operand types——*immediate*, *register* and *jump target*.

The node of possible exceptions is useful when *Generator* requires producing an instruction with some exception. For *ADDI*, its possible exception is *overflow*.

If one *ADDI* instruction is to be produced, the tree in Fig. 2 will be traversed. For the first and second operands, the type is *register* and then their value range is from 0 to 31. Similarly, the value of the third lies in the range from *2 ^15* to –(2^15 – 1).

The approach employed here gives enough power to generate useful and revealing test programs whilst keeping the complexity of the generator and the model reasonable. Moreover, the time needed for generation is kept within acceptable limits.

In addition, every instruction is attached a table to trace its testing coverage. The rows standard meaningful corner cases of the source operands and the columns list some interesting results. For example, the table of ALU instructions can be presented as Table 1.

Table 1. The coverage table for ALU instructions

Results / values	Zero	Overflow	Random Value
-1	True	True	True
0	True	True	True
1	True	True	True
Max.	True	True	True
Min,	True	True	True
Random value	True	True	True

When one combination is generated, the corresponding field will be set *TRUE*. So, the coverage of one instruction can be traced accurately.

3.2 Internal State Machines

FSM and *Resource* trace several kinds of internal states of the design processor.

3.2.1 Register States

There are 32 registers in our generator and each contains a data structure including the following fields.

- Its content
- The last instruction that accesses this register
- The register is accessed as a source operand or the destination
- The register is employed as a base address register or not

The reason to record field 2 and 3 is that they will be used to generate data dependence between instructions.

In addition, not all registers can be used as the base address register because it is possible that the memory address computed from any register and the immediate value does not lie in the range specified by given constraints. So, field 4 is employed to reduce the number of candidates.

Then, *Resource* module can handle the register allocation request based on the above structures. If it fails to find a proper one, some predefined instruction sequence can be inserted to solve the problem. For example, if no register can be used as the proper base address, one instruction sequence will be added automatically to load a right value into some register.

3.2.2 Cache States

Only D-Cache states are described because I-cache is read-only and its states can be handled as a sub-set of D-Cache.

One cache line contains the following fields: the tag, a 32-bit integer, four 32-bit data and the valid bits. All states of one line are described as follows and the FSM of D-Cache is described in Fig.3.

1. Start: The initial state and all cache lines are invalid.
2. Miss: The line is invalid but the preceding access to D-Cache hits.
3. Miss_miss: The line is invalid and the preceding access does miss.

4. Miss_hit: The line is valid but the preceding access does miss.
5. Hit_hit: The line is valid and the preceding access does hit.

When *Generator* requests producing a data hit instruction, it will locate all available cache lines based on valid bits and then get the set of valid destination addresses, which will be employed to select a proper base address register to generate valid load/store instructions. Both the selection of one valid cache line and the immediate value of the generated instruction may be random, so different load/store cases can be covered. If it is a data miss request, the similar operation will be executed except that all located lines are invalid.

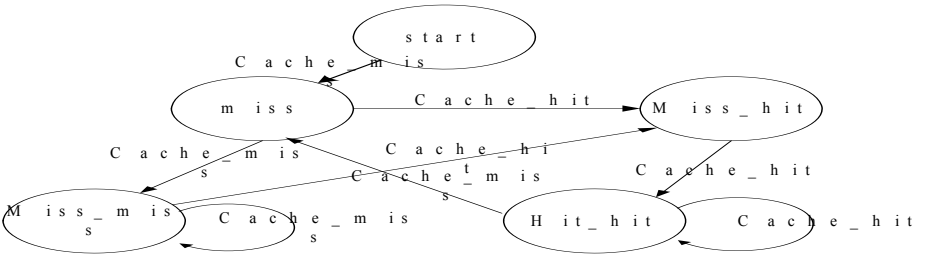


Fig. 3. FSM of D-Cache

3.2.3 States Between Pipeline Phases

Data dependence between instructions should be tested thoroughly because it is critical to validate the control logic of the target design, which is accomplished by the finite state machine described in this section.

Table 2. States between Pipeline Phases

State	Meaning
State_Null	The initial state
State_Ex_Mem	The latest instruction causes data dependence between EXE and MEM stages.
State_Ex_Aln	The latest instruction causes data dependence between EXE and ALN stages.
State_Ex_Wb	The latest instruction causes data dependence between EXE and WB stages.
State_ExMem_ExAln	The latest instruction causes two kinds of data dependence: EXE and MEM stages, EXE and ALN stages.
State_ExMem_ExWb	The latest instruction causes two kinds of data dependence: EXE and MEM stages, EXE and WB stages.
State_ExAln_ExWb	The latest instruction causes two kinds of data dependence: EXE and ALN stages, EXE and WB stages.

There are seven pipeline phases, IF, DE, RF, EXE, MEM, ALN and WB, in our CPU, and the possible data dependence between instructions is listed as follows.

- EXE and MEM
- EXE and ALN
- EXE and WB

Therefore, there are seven states in the FSM presented in Table 2 and transitions of these states are listed in Table 3.

Table 3. The transition of states

Constrains	The next State
Ex_Mem (It means to generate an instruction that will causes data dependence between EXE and MEM stages. And the following constrains have the similar requirement.)	State_Ex_Mem
Ex_Aln	State_Ex_Aln
Ex_Wb	State_Ex_Wb
Ex_Mem and Ex_Aln	State_ExMem_ExAln
Ex_Mem and Ex_Wb	State_ExMem_ExWb
Ex_Aln and Ex_Wb	State_ExAln_ExWb
Fail (It means that the constraint cannot be satisfied.)	The state is unchanged.

Any transition will be recorded in our generator, so it is easy to generate instruction sequence to introduce all possible data dependence.

3.3 Generation of Jump Instructions

There are three kinds of jump instructions in MIPS ISA, i.e., conditional jump, unconditional jump and unconditional register jump. To avoid an endless loop, different methods are designed for different backward jump instructions.

A new mechanism, *resource-locked,* is introduced for conditional jump instructions. That is, none of the instructions between the jump and the target will modify the jump condition except that a special one will be used to adjust the condition to control the number of loops. Another mechanism is designed for unconditional backward jump instructions, which inserts a conditional jump before the unconditional one to ensure that the loop will end in limited times. For unconditional register jumps, if the register content is less than *PC,* it can be regarded as a normal unconditional jump.

4 Co-verification Environment

We implemented a co-verification environment that contains the test program generator and the c-simulator, and its workflow is described in Fig.4.

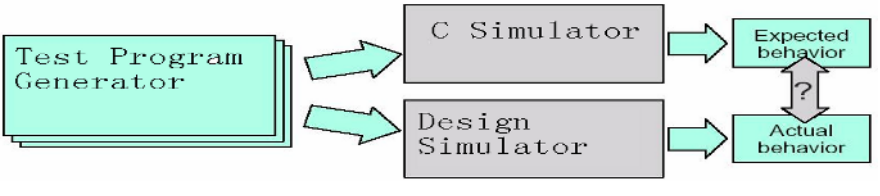


Fig. 4. The workflow of co-verification

At first, a user launches the environment, sets the constraints of test programs, and then the generator is started. When testing vectors are produced, both of c-simulator and the design simulator of the target CPU in different levels perform them, so testing traces generated can be compared directly to locate bugs if existing.

By the way, other vectors, including testing applications and OS, suitable for the c-simulator can also be executed directly by the design simulator in the co-verification environment.

5 Conclusion

This paper presented an approach to automatically test bench generation intended for simulation-based validation of systems. The approach exploits an accurate c-simulator to trace internal states, including memory access patterns, cache states, pipeline states and so on, of the target processor. That is to say, as soon as every vector is generated, it will be executed by the c-simulator and the generation policy will be tuned to produce the next vector with higher coverage. In contrast with the traditional methods that evaluate the testing coverage after generation, this method is more efficient.

In addition, we implemented a co-verification environment that contains the test program generator and the c-simulator. When testing vectors are produced under the environment, both of the c-simulator and the design simulator of the target CPU in different levels perform them. So testing traces generated can be compared directly to locate bugs if existed, which simplifies the verification process.

Reference

1. H.P. Sharangpani, M.L. Barton, “Statistical Analysis of Floating Point Flaw in the Pentium Processor”, Intel Corporation, 1994.
2. F. Casaubieilh et al., “Functional Verification Methodology of Chameleon Processor”, 33rd Design Automation Conference, Las Vegas, June 1996, pp. 421-426.
3. A. Aharon, D. Goodman, M. Levinger, Y. Lichtenstein, Y. Malka, C. Metzger, M. Molcho, and G. Shurek, “Test Program Generation for Functional Verification of PowerPC Processors in IBM”, 32nd Design Automation Conference, San Francisco, June 1995, pp. 279-285.
4. K. McMillan, Symbolic Model Checking, Kluwer, 1993.
5. R.P.Kurshan, computer-Aided Verification of Coordinating Processes: The Automatic-Theoretic Approach, Princeton Series in Computer Science,1995.
6. G.Al-Hayek, C.Robach: From design Validation to Hardware Testing: A Unified Approach, JETTA: The Journal of Electronic Testing, Kluwer, No.14, 1999, pp. 133-140.
7. B. Beizer, Software Testing Techniques, Van Nostrand Rheinold, New York, 1990.
8. S. Devadas, A. Ghosh, K. Keutzer: An Observability-Based Code Coverage Metric for Functional Verification. Proc. ICCAD’96.
9. P.A. Thaker, V.D. Agrawal, M.E. Zaghloul: Validation Vector Grade: A new Coverage Metric for Validation and Test, VTS’99: IEEE VLSI Test Symposium, 1999, pp. 182-188.

A New WCET Estimation Algorithm Based on Instruction Cache and Prefetching Combined Model

Guowei Wu and Lin Yao

Software College of Dalian University of Technology, Dalian 116023, China
{wgwdut, linyao}@dlut.edu.cn

Abstract. It is necessary to compute the execution time upper bound of embedded hard real-time program under the worst condition in embedded system design, which decides how hardware and software to partition and how to schedule process. Modern microprocessor which uses instruction cache memory and instruction pre-fetching increases the difficulty to compute the upper bound accurately. A new estimation method of embedded software performance based on instruction cache and pre-fetching model is proposed, which uses control flow graph and cache conflict graph and combine instruction pre-fetching into instruction cache analysis. It makes the execution time upper bound estimation under worst condition more accurate.

1 Introduction

It is an important task to predict the worst-case execution time (WCET) of a program in hard real-time systems. WCET analysis must be performed in order to guarantee that they will always meet the deadline of hard real-time system.

Until now, the analysis and estimation WCET is based on cache model. Liu and Lee [1] note that searching through all feasible program paths exhaustively is a sufficient condition for determining the exact worst case cache behavior. If there is a conditional statement inside a while loop for a program, this will become an intractable problem, unfortunately this happens frequently. Lim [2] expands timing schema methodology to incorporate instruction cache analysis, he also encounters the similar problem. To deal with this intractable problem, by proposing different pessimistic heuristics, the above researchers trade off instruction cache prediction accuracy for computational complexity. Robert [4] handles instruction cache performance analysis by using graph-coloring techniques. However, this approach has limited success even for small programs. All the above methods encounter computational complexity because they try to determine the exact sequence of instruction cache hits and misses. Different pessimistic methods are thus proposed to cope with this complexity [3], and they result in loose estimated WCET. Li [5] observes that the estimated WCET is only affected by the number of cache hits and cache misses. The actual sequence of hits and misses has no effect on the estimated WCET, so Li proposes an ILP estimation method which ignores other information and low computation complexity. Most of research work focuses on cache analysis during WCET estimation and ignores other properties of modern CPU, such as

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 557-562, 2005.

pipeline and instruction pre-fetching which increase the difficulty to compute the WCET and have important effect on WCET estimation precision. In this paper, we propose a new WCET estimation method which combines instruction pre-fetching into instruction cache analysis based on the work of Li [5].

In the following section, we describe the proposed estimation method, and in section 3, we evaluate the performance of the proposed method and give the experiment results, the conclusions and future work are presented in section 4.

2 Instruction Cache and Prefetching Combined WCET Estimation Algorithm

For analysis convenience, we define a new type of atomic structure, the line-block or simply l-block. An l-block is defined as a contiguous sequence of instructions within the same basic block that is mapped to the same line in the instruction cache. Thus all instructions within an l-block will always have the same cache hit/miss counts, and the same total execution counts [5]. Inside the basic block B_i, the l-block is denoted as B_{ij}, the cache hit and the cache miss counts of l-block B_{ij} are denoted as $x_{i,j}^{hit}$ and $x_{i,j}^{miss}$ respectively. The total execution time can be computed by summing the product of instruction counts by their corresponding instruction execution times. If there are N basic blocks, the total execution time (cost function) is given by:

$$T = \sum_{i}^{N} \sum_{j}^{n_i} \left(c_{i,j}^{hit} x_{i,j}^{hit} + c_{i,j}^{miss} x_{i,j}^{miss} \right) \quad (1)$$

Since l-block B_{ij} is inside the basic block B_i, its execution count is equal to x_i, thus $x_i = x_{i,j}^{hit} + x_{i,j}^{miss}$, here, $c_{i,j}^{hit}$ and $c_{i,j}^{miss}$ are the hit cost and the miss cost of the l-block B_{ij} respectively

The possible values of x_i are constrained by the possible values of the program variables and the program structure. The problem of finding the estimated WCET of a program will become an ILP problem if we can represent these constraints as linear inequalities.

The linear constraints of a program consist of two parts: program structural constraints and program functionality constraints. Program structural constraints are derived from the program's control flow graph (CFG), program functionality constraints are provided by the user to specify loop bounds and other path information.

For different programs, in order to analyze and give the constraints, all the constraints are passed to the ILP solver with the goal of maximizing cost function (1). The ILP solver will return the estimated WCET. We can incorporate instruction cache memory and pre-fetching analysis into cache model by modifying the cost function (1) and by adding a set of linear cache constraints and pre-fetching effect factor.

For any two l-blocks that are mapped to the same cache line, they conflict with each other if the execution of one l-block displaces the cache content of the other. Otherwise, they are called non-conflicting l-blocks. This leads to examine the control flow of the l-blocks mapped to that particular cache line by defining a cache conflict graph. Figure 1 is a cache conflict graph, it contains a start node *s* and an end node *e*.

If there exists a path in the CFG from basic block B_k to basic block B_m without passing through the basic blocks of any other l-blocks of the same cache line, a directed edge is drawn from node B_{kl} to node B_{mn}. For each edge from node B_{kl} to node B_{mn}, we assign a variable $p(k.l , m.n)$ to count the number of times that the control passes through that edge. We use B_{ij} and B_{uv} for general description. At each node B_{ij}, the sum of control flow going into the node must be equal to the sum of control flow leaving the node, and it must also be equal to the execution count of l-block B_{ij}. Therefore, two constraints are constructed at each node B_{ij}:

$$x_i = \sum_{u.v} p(u.v, i.j) = \sum_{u.v} p(i.j, u.v) \quad (2)$$

where u,v may also include the start node s and the end node e. Therefore, the contents of l-block B_{ij} are still in the cache. Every time the control follows the edge (B_{ij} B_{ij}) to reach node B_{ij}, it will result in a cache hit. Thus, there will be at least $p(i.j , i.j)$ cache hits for l-block B_{ij}. In addition, if both edges (B_{ij} ,e) and (s, B_{ij}) exist, then the contents of B_{ij} may already be in cache at the beginning of program execution as its content may be left by the previous program execution. Thus, variable $p(s , i.j)$ may also be counted as a cache hit. Hence,

$$p(i.j, i.j) \le x_{i,j}^{hit} \le p(s, i.j) + p(i.j, i.j) \quad (3)$$

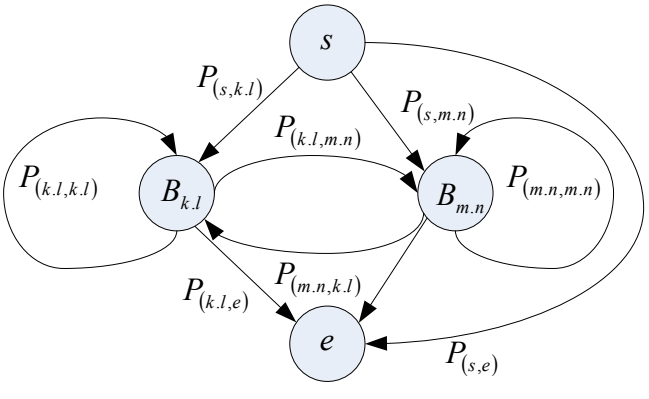


Fig. 1. Cache Conflict Graph

Otherwise, if any of edges (s , B_{ij}) and (B_{ij} , e) does not exist, then

$$x_{i,j}^{hit} = p(i.j, i.j) \quad (4)$$

Equations (2) through (4) are the possible cache constraints which don't include instruction pre-fetching effect. Next, we analyze a program to illustrate how to add the instruction pre-fetching effect into the cache model.

Figure 2 is the program flow graph, supposing that the program starts from B_2 basic block. $B_{6.3}$ and $B_{7.1}$ is non-conflicting 1-block, and they conflict with $B_{5.1}$. $B_{5.1}$ is pre-fetched by $B_{2.1}$. We modify the cache conflict graph: add the 1-block which includes instruction perfecting into the same cache set , in this example, it is $B_{2.1}$.

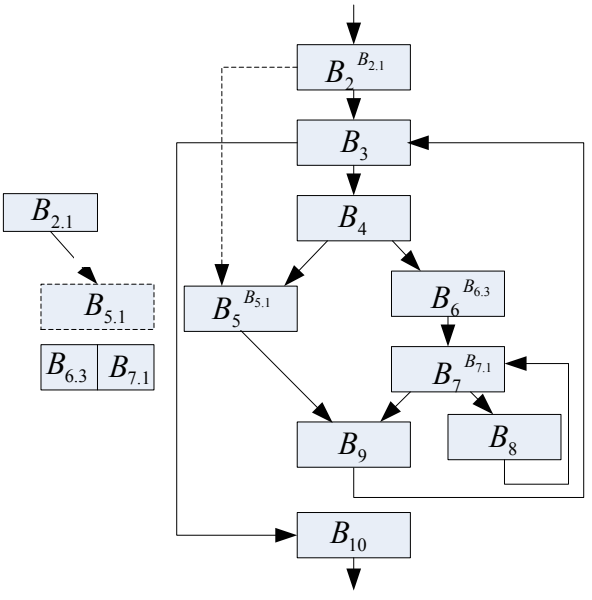


Fig. 2. Program Flow Graph

Pre-fetching node can be distinguished by attached property with other nodes, the new added node edge can be drawn according to the old method in cache conflict graph. For the pre-fetching node, when we judge the hit/miss property of its joined edge, we need to consider its pre-fetching data. In this example, $B_{5.1}$ is pre-fetched by $B_{2.1}$ and $B_{5.1}$ conflicts with $B_{6.3}$, thus mark the $B_{2.1}$ to $B_{5.1}$ edge as hit, $B_{2.1}$ to $B_{6.3}$ edge as miss. Thus, we can constrain the cache hit scope of 1-block as:

$$x_{5.1}^{hit} = p(5.1,5.1) + p(2.1,5.1) \quad (5)$$

Other instruction pre-fetching constraints information can be given according to this analysis way. Equations (2) through (5) are the possible cache constraints for bounding the cache hit/miss counts. These constraints, together with the functionality constraints and the structural constraints, are passed to the ILP solver with the goal of maximizing the cost function (1). Because of the instruction cache and instruction pre-fetching information are given, a tighter estimated WCET will be returned.

3 Experimental Results

To verify the proposed method, we use cache analysis tool Cinderella to estimate WCET of programs and evaluate the proposed method. Cinderella reads the subject program's executable code and constructs the CFG and the CCG, the user is asked to provide loop bounds and additional constraints information. The constraints are solved by the public domain ILP solver which uses the branch and bound procedure

to solve the ILP problem. Target platform is Motorola MMC2107 development board which contains a 40MHz Motorola MMC2107 processor, 128KB of main memory and several I/O peripherals. The M2107 processor contains an on-chip 8KB direct-mapped instruction cache organized as 32-16-byte lines.

Table 1. WCET estimation experimental result

Function	Measured WECT	Proposed method	Li's method
FFT	$1.25*10^6$	$1.23*10^6$	$1.18*10^6$
DES	$2.42*10^5$	$2.40*10^5$	$2.38*10^5$
Stats	$1.65*10^4$	$1.62*10^4$	$1.59*10^4$
DCT	$1.15*10^5$	$1.12*10^5$	$1.1*10^5$

For comparison convenience, we select the set of benchmark programs from [1] for our evaluation. The measured WCET in [1] is used as the actual WCET. We assume that the measured WCET of a program is very close to its actual WCET. We analyze every benchmark program and give constraints, then pass these constraints to ILP solver. Table 1 is the experimental results, where time unit is clock period counts, it shows that the proposed method gives a more accurate WCET estimation than Li's method. Combining instruction pre-fetching into instruction cache analysis can improve estimation precision, and the results prove that the model we give is right and feasible. We also compare computation complexity between the proposed method and other methods, Table 2 is comparison results, where time unit is second. It shows that the proposed method low the computation complexity.

Table 2. Computation complexity comparison result

Function	Liu's method	Proposed method	Li's method
FFT	15	0.08	0.075
DES	8	0.05	0.06
Stats	5	0.03	0.04
DCT	10	0.06	0.05

4 Conclusion

The properties of modern CPU, such as pipeline and instruction pre-fetching, have important effect on WCET estimation precision. Ignoring these properties during WCET estimation will result in loose WCET estimation. In this paper, we present a new method to find a tight bound on the worst case execution time of real time

embedded software. This approach combines instruction pre-fetching into instruction cache analysis. According to different programs, different program structural constraints and program functionality constraints which include cache and instruction pre-fetching factors are passed to the ILP solver with the goal of maximizing cost function. The ILP solver then returns a more accurate estimated WCET. Experimental results show that the estimated WCET is much closer to the measured WCET than that without instruction pre-fetching analysis. Experimental results also show that the proposed method lowers the computation complexity compared with other existed methods. Our next research work will focus on combining data cache and TLB cache analysis into instruction cache analysis, thus the WCET estimation result will get much more accurate.

References

1. Liu,J.C.,Lee,H.J.:Deterministic Upperbounds of Worst-case Execution Times of Cached Programs. In: Proceeding of the 15th IEEE Read-Time Systems Symposium,Vol.30,New York(1998)182-191.
2. Lim,S.S.,Young,H.B.,Gu,T.J.:An Accurate Worst Case Timing Analysis Technique for RISC Processors. In: Proceeding of the 15th IEEE Real-Time Systems Symposium,Vol.30,New York(1998)97-108.
3. Alan,C.S.:Reasoning about Time in Higher-level Language Software. IEEE Transactions on Software Engineering,Vol.15,No.7,pp.875-889,July 1999.
4. Robert, A.: Bounding Worst-case Instruction Cache Performance. In: Proceeding of the 15th IEEE Real-Time Systems Symposium,Vol.30, New York(1998)172-181.
5. Li,Y.S.,Malik,S.,Wolfe,A.: Cache Modeling for Real-Time Software Beyond Direct Mapped Instruction Caches. In: Proceeding of the 17th IEEE Real-Time Systems Symposium, Vol.35, New York(2002)35-42.

.

A Component-Based Model Integrated Framework for Embedded Software*

Wenzhi Chen, Cheng Xie, Jiaoying Shi

College of Computer Science, Zhejiang University, Hangzhou 312207, P.R.China
wzchen@cad.zju.edu.cn,
arthurxie@vip.sina.com, jyshi@cad.zju.edu.cn

Abstract. The development of distributed, concurrent software in embedded systems is becoming increasingly complex and error-prone. Model-based integration of reusable components is advocated as the method of choice. To this end, we propose a framework to support component-based model integration, hierarchical functionality composition, and reconfiguration of systems with continuous and discrete dynamics. In this framework, components are designed and used as building blocks for integration, each of which is modeled with abstract ports, reactions, and communication schemes. It uses hierarchical composition to hide the implementation details of components, and keeps the components at the same level of hierarchy interacting under a well-defined model of computation. Code generation takes the design decisions down to the final running system. Within this framework, embedded software can be constructed by selecting and then connecting components in a functionality repository, specifying models and transforming them to executable codes.

1 Introduction

The model-based approach has proven to be effective for fast and low-cost design and simulation of embedded systems, such as automotive systems. However, due to the lack of a common framework, the benefits of model-based approach are limited by the manual process of extracting information in one model for reuse in another. Furthermore, the current practice in embedded software development relies heavily on ad-hoc implementation to meet the various constraints of the underlying platform. Although component-based software development and integration are known to be efficient for software reusability, such an approach is neither well-defined nor well-understood in the embedded system domain.

* This work was supported in part by the Hi-Tech Research and Development Program of China (863 Program) under Component-based Embedded Operating System and Developing Environment (No.2004AA1Z2050), and Embedded Software Platform for Ethernet Switch (No. 2003AA1Z2160); In part by the Science and Technology Program of Zhejiang province under Novel Distributed and Real-time Embedded Software Platform (No. 2004C21059).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 563-569, 2005.

Embedded systems are intrinsically heterogeneous. It consists of various device drivers and various control algorithms, which usually exist as software components. The physical processes to be controlled are usually continuous but the algorithms are implemented using discrete software components. There are hybrid models that match different parts of a system, for example, continuous time(CT) models for ordinary differential equations, finite state machine(FSM) models for plant operations, discrete event(DE) models for network communication, and synchronous data flow(SDF) models for signal processing. Although each individual model is relatively well-understood, it is difficult and complex to implement the integration of heterogeneous models.

An effective solution is to construct a common component-based framework and use it for model integration. In this paper, we present a framework that supports the component-based model integration and implementation process. The framework provides a component repository and hierarchical models, and can be used to specify software structure, distributed functionality, and system constraints. Function definitions of a component are separated from non-functional aspects, especially timing and resource constraints. Components can be structurally integrated via their communication ports, through which the state transitions of the system can trigger reactions. The functionality of a component can be implemented using a different model and enables reconfiguration after structural composition. The framework provides a clean way to integrate different models by hierarchically composing heterogeneous components. This hierarchical composition allows one to manage the complexity of a design by information hiding and component reuse. The framework has been applied to the Pcanel operating system that we developed at Zhejiang University for cybernetic transport system. The model integration framework allows seamless composition of vehicle applications with distributed real-time functionality to enforce desired efficiency and safety.

2 Component Repository

A component-based embedded software design is modeled as a set of software components and their interactions. Components are pre-defined software modules and treated as building blocks in integration. The integrated embedded software can be viewed as a collection of communicating reusable components.

The component repository contains the core software components for reusability and integrated descriptions about hardware and bus systems. The characteristics of the software components are also stored in the repository, e.g. test case, code size and worst case execution time. Interfaces must be part of the repository. In distributed embedded systems, the communication among software components can take place by data buses or internally on the processor.

The interfaces of the software components are defined globally. A formal notion of component interface provides a way to describe the interaction between components, and to verify the compatibility between components automatically. The theory of timed interfaces [1] is used to specify both the timing of the inputs a component expect from the environment, and the timing of the outputs it can produce. The

formalism of resource interfaces [4] is used to specify component interfaces that expose component requirements on limited resources.

The component structure defines the required information for components to cooperate with others in a system. Execution profiles define the execution environment or infrastructure of a component. Examples include scheduling policies, real-time constraints and resource demands. A component can be customized for use in different environments by selecting different execution profiles. Components have a collection of abstract input/output ports. Ports are shared states that allow components to communicate with each other via tokens. The number of ports needed for a component can be determined and customized by the system integrator. Different types of ports with different execution profiles can be selected to achieve different performance requirements.

Reactions define the functionality of the component that can be invoked outside the component. In our model, reactions are represented as a set of triggers with actions. Triggers are guards of some meaningful system states, such as time, signals, and events. A component with other forms of reactions, such as function calls, can be integrated into the system by mapping each of them to a unique trigger. Using triggers enables actions to be scheduled and ordered adaptively in distributed and concurrent system, and enables components from different vendors to be integrated into the system without the source code modification. With such a component model, the system can be designed by connecting cooperating components through their ports, and the system execution can be done by having external state transitions like timer interrupts or sensors trigger a sequence of reactions in components.

The semantics of reaction is designed to separate function definitions from state transition specifications, and support reconfiguration. Reactions of a component are specified in a table form [7]. When a trigger is activated at runtime, actions are invoked according to the state table. The table enables the control logic to be reused, and enables remote or runtime reconfiguration. The state table can be treated simply as data and passed around the system. This compactness of table is useful for embedded systems with limited resources and distributed environments, such as in-vehicle control systems. Figure 1 shows a component-based design for continuous time(CT) model and the component structure of corresponding implementation.

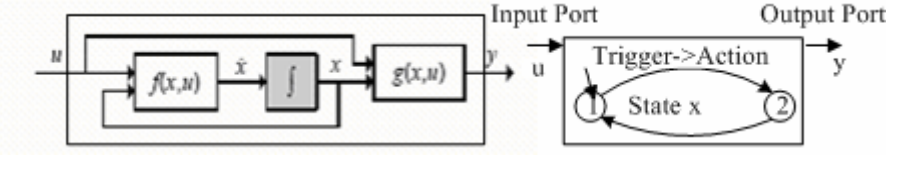


Fig. 1. Component-based design and implementation

Component connection network(CCN) combines the software components with each other. The framework supports hierarchical composition to keep the systemic view. The communication among components is carried out on token basis. The token flows are scheduled within models. In a hybrid system, hierarchical heterogeneous models cooperatively direct the token flows. Based on the CCN, token flow network is constructed to analysis and verify concurrent and real-time functionality of complete embedded software.

The complete software can not independently from the hardware. The execution of software depends on the underlying processor architecture, memory mapping, data bus, or device register. For reuse of components, hardware platform descriptions are also stored in the repository.

3 Distributed Functionality

More and more embedded systems consist of a network of electronic control units (ECU) connected via a bus. As the platform architecture shown in Fig. 2, each ECU consists of the controller, an operating system, a dedicated communication layer, and one or many application reactions. The functionality of a component is modeled as component structure. The functionality of a system is modeled as component connection network. The communication among components is modeled as token flow network. The distribution of functionality among ECUs is transparent in a high-level systemic view. The communication between reactions of spatially separated ECUs is wrapped by the communication layer. The integrated system model may span hybrid bus systems, such as Controller Area Network and Local Interconnect Network.

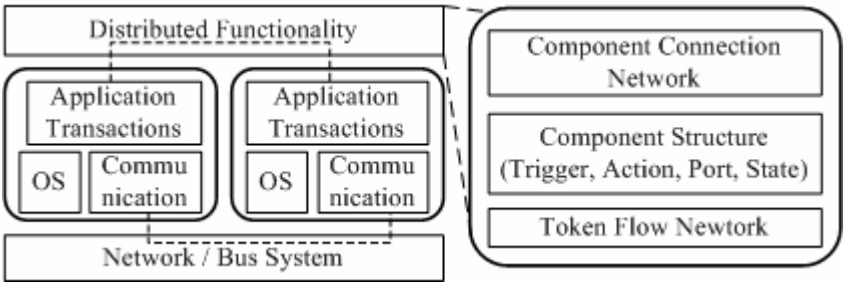


Fig. 2. Platform architecture

The composition model defines how software can be integrated with given components. Since each reusable component is implemented with a set of reactions that uniquely define its functionality, components can be selected based on the match of their reactions and design specifications. The integration of reusable components can be viewed as linking the components with their reactions.

A composite is an integration of reusable components. The model of the composite links the components with their reactions, allowing for the observation and manipulation of the runtime states and behaviors internal of components. Furthermore, to facilitate modularity, a composite itself, together with the components within its model, can be treated as a integrated component at a higher level of hierarchy, which means that a composite can be encapsulated to a component. The member component behaviors determine the reactions and states of the integrated component. When applied formal models, the composite maintains assurance of diversified non-functional aspect, such as timing and deadlock.

Models are independent of implementation of components. Thus, Reusable components in integrated software are organized hierarchically to support integration with different models. A complete system configuration is a set of hierarchical compositions of models and reusable components.

4 Code Generation

The integrated software obtained from the composition model cannot be executed directly on a platform since the composition model only deals with distributed functionality. Code generation approach is a migration path from design-time models to runtime models. A typical code generation process assumes a flat operating system support and generates a stand-alone program that is then compiled into an application. Our framework provides a runtime system natively supporting executable models and distributed deployment. It greatly helps code generation and improves the quality of final software. The runtime system can utilize hardware support (such as SMP) and communication systems (such as CAN). In addition, there are certain assumptions, like resource reservation and timing predictability, can only be achieved by OS-level runtime systems, but not easily by stand-along programs.

To obtain and deploy complete software, components have to be transformed to reactions, which are basic schedulable units of the runtime infrastructure. A reaction is synthesized by code generator from a sequence of actions associated with an external trigger, which represents physical process such as interrupt, signal, and event. The code generator generates a runtime implementation that consists of a network of computing blocks communicating through a publisher-subscriber service.

A synthesized reaction has access rights on internal states of all components that own the actions. Such access rights are constructed during reaction initialization to avoid concurrent data competition. On arrival of a trigger, the reaction executes the pre-compiled actions in static order. The reaction is not reentrant, that in each round of a reaction execution, exactly one trigger is processed.

All reactions execute with statically assigned priorities in the runtime model. A reaction with high priority preempts lower-priority reactions. Actions within a reaction are executed at the same priority assigned to the reaction itself. The execution sequence of actions modeled at design-time to achieve functionality is preserved at runtime. Compared to other models requiring dynamic priority assignment [5], our implementation has low runtime overhead, lesser complexity and better support for massive concurrency.

5 Related Work

Since most embedded systems deal with safety-critical applications, model-based design and formal analysis are highly desired and widely used in software development. Ren et al developed an approach based on the Actor model for distributed real-time systems [6]. An Integrated Object-Oriented Environment is proposed for Real-Time Industrial Automation Systems [2]. Stewart et al used port-

based objects to support dynamic reconfigurable real-time software [3]. All of these frameworks agree on modeling the components as autonomous self-contained software modules and using event mechanisms to describe the connection of components. However, Most of the previous research in the literature has focused on component model, while largely ignoring the heterogeneous properties of software for hybrid systems. Systematically integrating heterogeneous components is crucial to design complex embedded systems. It is difficult for engineers to reconfigure and analyze components and their integration. Lack of context and environment descriptions may further introduce mismatching problems of architecture and interface inconsistency. Out framework is inspired by practical applications like automotive control applications which are model heterogeneity. The component-based model integrated framework we proposed is designed for re-usability of components with model heterogeneity.

6 Conclusion

In this paper, we presented a component-based model integrated framework for embedded software. A reusable component in our framework is modeled with communication ports, triggers, and reactions for separate functionality specification and reconfiguration. Component repository contains components for reusability and integrated descriptions for executing environment adaptation. Distributed functionality within hybrid models is designed by hierarchical composition of components. Code generator transforms the design to implementation by OS-level runtime support. Such a framework enables multi-granularity and vendor-neutral component integration, as well as functionality reconfiguration. Our future work will focus on the timing and resource analysis for integrated components. The framework presented in this paper makes it possible to separate the timing and resource analyses from the functional integration.

References

1. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Marielle Stoelinga. Resource interfaces. Proceedings of the Third International Conference on Embedded Software (EMSOFT), Lecture Notes in Computer Science, Springer-Verlag, 2003.
2. Becker, L. B., Gergeleit, M., Nett, E., Pereira, ., C. E., An Integrated Environment for the Complete Development Cycle of an Object-Oriented Distributed Real-Time System, 2nd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'99), Saint-Malo, France, pp. 165-171, May 1999.
3. D. Stewart and P. Khosla, "The chimera methodology: Designing dynamically reconfigurable and reusable real-time software using port-based objects," International Journal of Software Engineering and Knowledge Engineering, vol. 6, no. 2, pp. 249-277, 1996.
4. L. de Alfaro, T.A. Henzinger, and M.I.A. Stoelinga. Timed interfaces. In Embedded Software, Lect. Notes in Comp. Sci. 2491, pages 108-122. Springer, 2002.

5. M. Saksena, P. Karvelas, and Y. Wang. Automatic synthesis of multi-tasking implementations from real-time objectoriented models. International Symposium on Object-Oriented Real-Time Distributed Computing, March 2000.
6. S. Ren and G. Agah, "A modular approach for programming distributed real-time systems," in Lectures on Embedded Systems: Eur. Educational Forum School on Embedded Systems (LNCS 1494), Veldhovan, The Netherlands, Nov. 1996, pp. 171-207.
7. T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Synthesis of FSMs: Logic Optimization. Kluwer Academic Publishers, 1997.

A Cooperative Web Framework of Jini into OSGi-based Open Home Gateway

Zhang-Long Chen[1], Wie Liu[1], Shi-Liang Tu[1], and Wie Du[2]

[1]Department of Computer Science and Engineering, Fudan University, Shanghai
{chenzl, wliu, sltu}@fudan.edu.cn
[2]College of Management, University of Shanghai for Science and Technology, Shanghai

Abstract. The administration of heterogeneous networks with many embedded equipments and mobile devices is a hard and time-consuming task. Today's methods only provide static configuration files and make the addition and removal of devices a manual chore. The Open Services Gateway Initiative (OSGi) specification defines a service-oriented framework for use in residential gateways. OSGi framework acts as a gateway from the Internet to consumer devices attached to the residence's home-area network. And Jini is an infrastructure for spontaneous and ad hoc service networks. It allows users to find services which consumer devices provide without prior knowledge of their network environments. The paper presents a cooperative web framework for the integration of Jini into OSGi open home gateway which can provide much easier, more intelligent and powerful home network control service. It can federate home-area network devices and services in secure ways by Internet. This paper puts forward relative design and reference implementation.

1 Introduction

There is a rapidly growing market for intelligent mobile devices and embedded equipments. Devices such as Personal Digital Assistant (PDA), mobile phones and embedded equipments at home are getting cheaper, smaller and more powerful. The uses of mobile embedded devices is moving geographically, connecting the device often to different local networks in order to finish different tasks and maybe providing different services and control interfaces to users by Internet[1].

OSGi is making developers and enterprises realize the potential of the consumer equipments market such as virtual intelligent home and intelligent home health care [1]. The OSGi specification defines a service platform [2] that includes a minimal component like model and a small framework for managing the components, including a packaging and delivery format. The Jini technology is also a service-oriented approach to realize spontaneous networking. It addresses the challenges of spontaneous distributed systems such as robustness, self-healing, administration-freeness and heterogeneity. But how to provide the impromptu and federative

This work is supported by China "863" high technology research software project fund (2003AA1Z1120).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 570-575, 2005.

management of the embedded and mobile devices in heterogeneous network environments is a problem.

To alleviate the lack of flexibility and transparency of reconfiguration for the user when changing the network environment, the paper proposes a cooperative web framework for the integration of Jini into OSGi open home gateway. And it presents related design and reference implementation. In the rest of paper, it analyses the OSGi architecture, Jini technology and the cooperative web framework. Section 2 describes the characteristics and drawbacks of OSGi technology. Section 3 puts forward the cooperative web framework for the integration of Jini into OSGi open home gateway and Section 4 is related analysis and reference implementation. Section 5 is conclusions and future work.

2 The Characteristics and Drawbacks of Current OSGi Technology

The key aspects of the OSGi mission are multiple service, wide-area networks, local networks and devices. The central component of the OSGi specification is the service gateway that acts as the platform for many communication-based services. The service gateway can enable, consolidate and manage voice, data, internet and multimedia communications from the home, office and other locations. The OSGi framework creates a host environment for managing bundles and services, while a bundle is the physical unit of deployment in OSGi and a logical concept used by the framework for organizing its internal state. A bundle is a Java JAR file that contains a manifest and some combination of Java class files, native code, and any associated resources. An installed bundle in the framework is uniquely identifiable by either its bundle identifier, a number assigned dynamically by the framework when the bundle is installed, or by its location.

The trend of the electronic market is taking aided by the concomitant development of mobile embedded devices, and it suggests a major change from the way electronic commerce is done today. The increased use of PDA and laptops has shown a new horizon of proliferation in the electronic market, since e-commerce and e-home services and transactions processing facilities need to be accessed from a wireless or wired device. Discovering services dynamically will become increasingly important in the heterogeneous networks. Some examples of resource discovery protocols, which allow the devices to find each other and perhaps to talk each other, are also available in the market. OSGi aims to define and promote open specifications for the delivery of multiple services over wide-area networks to local networks and devices. The current specification of OSGi is 3.0. But it doesn't present rational solutions to the impromptu and federative management of the embedded equipments and mobile devices in heterogeneous network environments. Today, however, the change of network surroundings is linked to a number of reconfiguration tasks to be done manually. There is an obvious gap of transparency between the need to change the network environments on the one hand and the methods to support this change in a transparent and convenient manner for the user on the other hand.

3 A Cooperative Web Framework of Jini into OSGi Open Home Gateway

3.1 Software Solution: The Integration of Jini into OSGi Open Home Gateway

In the normal Jini scenario, when both a client and service are within a LAN (local-area network), the client downloads a "proxy object" which it can use to control the service. Initially it hopes to be able to keep up to use this Jini architecture to control Jini services over the Internet using applets by allowing the applet to download a service's proxy object and directly contact the service's machine. However, when the client is outside the LAN this same interaction raises problems. At first, the client would need to install the core Jini files. Moreover, the use of multicast to discover the lookup service cannot be used. This however can be solved by Jini's unicast lookup ability where the client specifies a known IP address of a lookup service. Unfortunately there are more severe problems. Even if the client knows the IP address of the machine running the lookup service and uses Jini's unicast facilities to try to contact it, it is unlikely to get through if that machine is behind a firewall. If the firewall is configured to permit it to access to that relative port on the lookup service machine, then the client will be able to access it and download the proxy object. Even if having to open ports for every machine it runs a service on the communication between the proxy object and the service back-end. RMI will be often used.

One solution to this problem is to try to tunnel RMI over HTTP. This need encapsulate the client RMI call in an HTTP POST request and unpack it on the server side. But there is extensive negative experience with this on the RMI users mailing list. This issue has been raised previously as the Jini attempts to access Jini services over the Internet. Moreover, at the time of writing, the RMI over HTTP solution is referenced as not working. Even if this solution becomes feasible, the client would have to install the core Jini files in advance. This method to these this problem is to add an extra level of indirection to the standard Jini architecture.

3.2 Reflection – A Generic Jini Client

For a Jini client, once the lookup service has been found, it can be searched for services that implement a specific interface. Therefore, if the client is to search for a specific type of service, it must be aware of that service's interface at compile time. A client can however search for all services available implementing any interface by passing null as a parameter in the methods that search the lookup service. In cases where the client is automated, this is not a problem because it doesn't make sense for programmatic clients to call a service they know nothing about. However if there is human interaction with the client, it may be possible to provide the user with enough information about the service at run time so that user can sensibly call the service. Although the client doesn't know any information about the service, it aims to extract enough information about it at run-time and provide this to the user so that user can make judgment on whether or not it makes sense to access the service. Java's "Reflection" abilities permit it to discover the type of an object, the interfaces it

implements, and the methods those interfaces define when it gets the object at run time. These capabilities can be used to augment the web link architecture.

The first stage supplies all discovered services, their class names and a description of each service. The user then chooses a specific service. This selection is passed to the second stage, which provides a list of all the interfaces supported by the selected service. The user then selects a specific interface. This selection is passed to the third stage, which can offer a list of all the methods provided by that interface. The user selects a method to invoke. The method is called on the service and the final stage shows the result.

3.3 Requirements and Solution of the Cooperative Web Framework Security

Currently the specification of OSGi comprises many service, such as remote management reference and Http service. And every idiographic service may have security requirements and specification. So the OSGi specification doesn't present any implementation about security. However, the security problems that are bound to be present in any large-scale deployment of Jini are not adequately addressed by either the current revisions of Jini technology or the underlying Java security solutions. Therefore it analyzes the security requirements of OSGi and Jini in different environments. Security threats and high level security goals are identified, and the implementation of these high level goals using lower level security mechanisms is described. Based on the identified requirements, the architecture for the cooperative web framework of OSGi and Jini security is proposed. The architecture is based on the trust management approach, and uses Simple Public Key Infrastructure (SPKI) certificates for authorization.

Traditionally, security has been grounded on identity authentication and locally stored access control list (ACL). This has been the case even in distributed systems. However, that approach has a number of shortages, for example, the problem of protecting the operations that are needed for managing access control list remotely. In Blaze et al. argue that "the use of identity-based public key systems in conjunction with ACL is inadequate solutions to distributed system-security problems." Examples of trust management systems include the Policy-Maker, which originally introduced the term trust management [4], its continuations KeyNote and KeyNote2 [5], and in some respects, SPKI [6] and its applications, including TeSSA.

The relationships form two loops. The service wants to verify that the user is authorized to use this particular service, and the user hopes to validate that the client is talking to the accurate service. Secure system signs any piece of data using the proxy's key. Secondly, the proxy can request some permission to be delegated from the user to the proxy's public key This delegation is expressed as a SPKI certificate and the certificate is given to the proxy. The security manager can offer one service for the client applications. Given a proxy instance, a client application can ask for the public key of the corresponding service. The application can implement an authentication user interface, which could use, for example, name certificates given by the proxy for authentication.

4 Related Implementation

4.1 A SOAP Jini Client/Web Bridge

SOAP is an XML based distributed programming specification. Implementations of this specification are available in many languages. Since Jini services and clients are built in the Java programming language, an implementation of SOAP in Java is necessary in order to uncover the Jini services as SOAP services that can be accessed by SOAP clients over the Internet. Apache-SOAP is one such implementation and can be downloaded from the Apache web site. Apache SOAP can be used as a client-side library to invoke SOAP services available remotely, or as a server-side tool to expose SOAP services. The configuration of the client and server are both documented with the Apache-SOAP download. From the server's point of view, among other things, it needs a web application server that supports servlets and JSPs (Java Server Pages). And the OSGi specification comprises Http service, Servelet service etc.

In order to act as a Jini client and contact any Jini services on the LAN, the OSGi servlet engine must be modified. Once we set up the servlet engine configured for Apache-SOAP, we can register a Jini client as a SOAP service. The process of registering a SOAP service is well documented in the Apache-SOAP download.

The SOAP message can then be created and the specific method it attempts to invoke can be passed to it. Using SOAP, and the modified Jini web link architecture as the means of invoking Jini services avoids the problems of firewalls and of needing to install the core Jini technology files on the client. However in order to invoke Jini services via SOAP over the Internet, the Internet client (or SOAP client) must still install the SOAP implementation files as well as an XML parser. Moreover, the SOAP client is invoked using a command line interface rather than via a browser. Finally, the speed of a SOAP client-server interaction is quite slow. This is due to the fact that XML has to be parsed both on the client and on the server machines.

4.2 A Servlet Jini Client/Web Bridge

The advantages of using a series of Java servlets as the Jini "client/web link" are apparent. Firstly, the speed advantage of servlets over SOAP is gotten, not only because of the necessity of parsing XML at both the SOAP client and server, but also because of the multi-thread capabilities of servlets. Furthermore, the obvious advantage of servlets over SOAP in this situation is that the Internet client can be a simple HTML page thus not needing any distribution or installation of client side files. Finally, using servlets allows easy and clean use of Java's reflection capabilities to develop a generic Jini client that can control all Jini services without any advanced knowledge of them. While these capabilities can be used with a SOAP implementation in Java, the clean well defined model for processing data transfer between servlets simplifies this process. As in the previous SOAP service scenario, OSGi-based gateway is located on the client/web bridge machine in the web link architecture.

4.3 Implementing a Generic Jini Client by Reflection

As described previously, Java's "Reflection" capabilities allow it to discover the type of an object, the interfaces it implements, and the methods those interfaces have when system gets the object at run time. These abilities can be used to augment the web link architecture to create a generic Jini client. This Jini client should be able to access all Jini services, even ones that are created after the client has been written.

The four-stage addition to the web bridging architecture used to create the generic client is described in above. These four stages can be implemented by providing four servlets which can pass the user's choice at each stage to the next servlet. In the first stage of the generic client, the user selects which service he/she would like to access. This selection is passed onto the second servlet. The second servlet then displays the interfaces implemented by the selected class. The user then selects an interface and that selection is passed to the third servlet. This servlet shows all the methods defined by the selected interface.

5 Conclusions and Future Work

The OSGi services gateway enables the connectivity and management of several devices, including set-top boxes, routers, alarm systems, cable modems, energy management systems, consumer electronics, PCs, and the residential gateway. And Jini is a valuable technology for the integration of embedded and mobile devices into heterogeneous networks, at least for stateless services or services which don't depend on long term network connections such as telnet or ftp. But how to integrate this promising technology for providing reliable services is a stringent problem. The paper puts forward a cooperative web framework which makes Jini and OSGi architecture work cooperatively. And it presents the security requirements of the cooperative framework and offers possible solution. In the future it is planning to define a standard API for secure services. And it needs to evaluate the performance of the cooperative framework and secure services capability.

References

1. Open Services Gateway Initiative, http://www.osgi.org, 2003
2. Open Services Gateway Initiative, "OSGI Service Platform," Release 3, 2003.
3. Sun Microsystems, "Jini Specifications
4. Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 164–173, California, May 1996.
5. Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system version 2. RFC 2704, IETF, September 1999.
6. Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylönen. SPKI certificate theory. RFC 2693, IETF, September 1999.

A Structure Modeling Method for Multi-task Embedded Software Design

Jiamei Cai, Tieming Chen, and Liying Zhu

College of Software, Zhejiang University of Technology,
310014, Hangzhou, China
{cjm, tmchen}@zjut.edu.cn

Abstract. Development of multi-task embedded software involves very complex modeling process. Based on the data flow diagram (DFD) method, a structure modeling method for multi-task embedded system design is present in this paper. This structure modeling approach mainly depends on the task-based DFD method. As a concrete instance, the development of a project on IMS detector design is discussed using above method. It is true that DFD-based structure model is an effective solution for some real time embedded system design.

1 Introduction

It's very useful to establish a reasonable software structure at the beginning of the design for multi-task embedded system. So far, there exist some methods for presenting embedded real time system, which include Language Presentation and Mathematical Analysis, Flow Diagram, Structure Diagram, Finite State Machine, and Data Flow Diagram, etc. Among all above methods, Data Flow Diagram is the only one that can express the system structure characteristics and has the ability of describing parallel procedure, and is most suitable to embedded software structure design. Therefore, it's feasible to use task-based DFD as the analysis tool for real-time multi-task embedded system development.

We do system requirement analysis and functionality modules definition at first, then draw the DFD according to the dada flow analysis of functionality modules, with the purpose to obtain the task DFD under restricted conditions. The asynchronism of system functionalities and transformation of DFD are then analyzed to make sure which part is parallel or sequential in procedure. All tasks are specified in order to achieve the task-based DFD finally.

We denote circle angle rectangles as tasks, oriented edges as data flows and events between tasks, and such task-based DFD can just properly identify various tasks' either dependent or restricted relationships based on data transmit. Fig.1 is an illustration of DFD with event constraint where A and B are the subtasks of task C.

From the illustration we know that attributions and actions of tasks are well enveloped, and tasks communicate with others or external environments only through data input, data output, event input and event output. Such DFD under event constraints here can model the embedded software system structure well.

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 576-581, 2005.

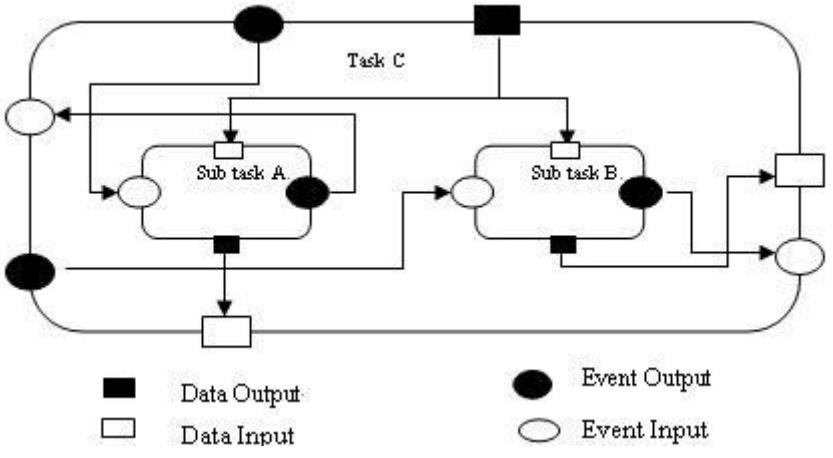


Fig.1. Illustration of DFD under Event Constraint

2 Task Constructions and Task-Based DFD Modeling

In real-time operation system, task is the smallest unit competing for system resource and is independently running in parallel. The task's outstanding feature is holding the private data container, for example, the defined stack during a task being initiated preserves task execution trace. So constructing system application tasks in correct way is critical for system coordination and simplification, as well as affects system performance, real time level and system throughput, etc.

In order to build system basic task DFD model, tasks are first constructed from considering several system aspects: I/O functionality, internal functionality, task cohesion, and task priority [1,3].

2.1 I/O Task Construction

In general, the transformation performance relies not only on the running efficiency but also on I/O device, so tasks about I/O can be independently constructed. The construction rule is that only system tasks related to I/O devices are constructed and further implemented in code, with the performance only limited to I/O devices throughput not to CPU.

2.2 Internal Task Construction

Internal functionality is the internal event functionality, which controls, coordinates or processes other system functionalities. Internal functionalities can be assigned into various tasks after internal tasks construction. According to control type, internal

functionalities may be constructed as periodic tasks, asynchronous tasks, control tasks and user interface tasks, etc.

2.3 Task Cohesion and Coupling

From the view of software engineering, task cohesion can simplify system module structure and reduce system expenditure. Anyway, excessive cohesions may result in hard management of task priority, blocking and other real time parameters. In addition, the data communication amount should be as concise as possible to avoid control cohesion. If control cohesion occurs, some corresponding measures should be taken to keep synchronization or exclusion, and to avoid bound resource collusions.

2.4 Task Priority

Prior task constructs based on priority division, which can balance CPU occupied time. If there exists functionality that needs large computing resource, it should be set to lower priority in order to be alternated momentarily by higher priority tasks. Thus the higher priority tasks, which occupy CPU less time, can always be kept on running while the above computing tasks seem to only consume idle CPU time.

2.5 Structure Modeling Design Procedure

First of all, basic tasks DFD is constructed, then it is optimized through tasks partition, simplification, mergence, etc. based on the relationship between data input and output, event input and output, as well as some constrain conditions. Successively new tasks DFD is combined. We can then obtain the final number of tasks due to the new tasks DFD, and start system concrete design depending on task modules.

3 Development of IMS Detector Embedded Software

3.1 System Principles in Brief

IMS (Ion Mobility Spectrometry) is an electronic detector device, which can identify whether some chemical ingredients contained in the given liquid through measuring distance-given mobility time of the ion in the given electromagnetic fields under the circumstance of given temperature and gas pressure. The distance mobility rate is distinguished based on the fact that different substances have different structure features, so the mobility time is also unique [2]. In the hardware structure of IMS, IMC (Ion Mobility Tube) is the most important component which structure is illustrated as Fig.2.

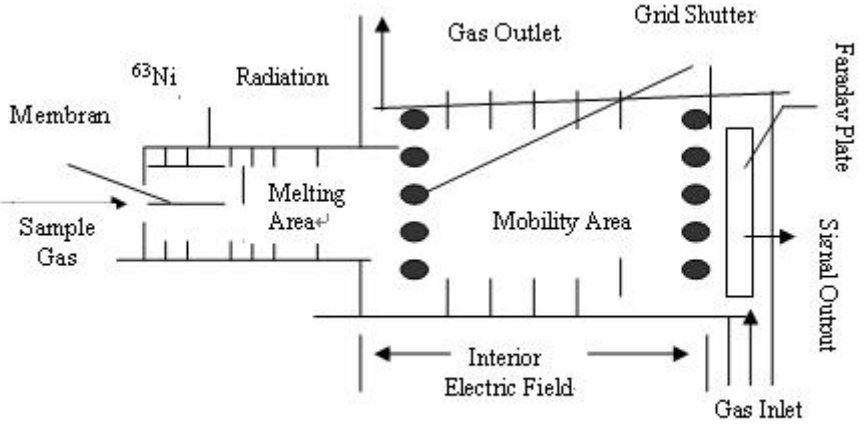


Fig.2 The Basic Structure of IMC

3.2 Software Functionality Modules

According to the working principal of IMS, we conclude the software running procedure as following. Once the system starts, various tubes should be heated up to a given fixed temperature. And the ion signals are continuously gathered to appear on LCD screen as EID. During the system running, the motor, pump and valve should all be controlled in proper manner while the control analyzing tube is heated up to a given temperate. Finally, all useful information such as detect status, EID, detect result, etc. will be shown on screen respectively.

We then define six key functionality modules: Real time data gathering module Tdg, real time control module Tc, display module Td, key-press process module Tk, analyzing module Ta and database module Tdb.

We can obtain the basic tasks DFD from above analysis, also we can establish the constrain relationship between data and events of tasks as shown in Fig.3

3.3 Tasks Partition

Basic tasks DFD construction is the initial step for functionality modules partition. More detailed and reasonable tasks partition should be taken for simplifying complex relationship between tasks. Here we introduce the next steps of tasks partition for IMS software based on partition rules in section 2.

(1) Due to the different sampling periods, we decompose data gathering task into two independent tasks: Channel data gathering task Tmc and Ion signal gathering task Tion, with two input event e_{ht} and e_g, two output data d_m and d_{ion} respectively, also the output event e_e for Tion. Note that d_{ion} is stored in non-cache region.

(2) Task Td and Ta use d_{ion} while d_{ion} is not in cache but its sampling period is very short. So it's necessary for Tion to decompose a new task called ion signal data transmitting task Tmv, and move d_{ion} from non-cache to cache.

(3) Task Td involves lots of functionalities and is really complicated, so it may be partitioned into the main task Tm and interface task Tui. Furthermore, Tui is not bounded with low priority, so it forks a new task called Tal, whose priority is higher than Tui, to correspond to warning task

(4) Control task and channel data gathering task have the same time period, so we can incorporate them into a new task Tmcc based on the time cohesion rule.

(5) Because the database task connects to PC through the serial port, we consider it as an independent task according to I/O task construction rule.

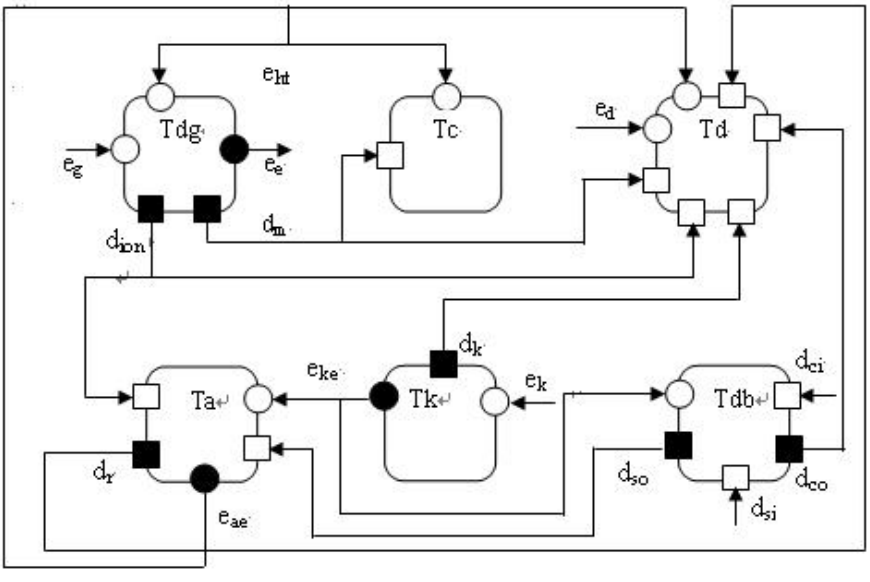


Fig.3. Basic Task DFD

Once a new fractionized tasks set been created after above partitions, a new tasks DFD model outcomes which is shown as Fig.4.

4 Conclusion

Single-task environment development method has not suited for the embedded system because such software needs not only the processing ability for large real-time data, but also the function for controlling multi-object parameters, which are always non-linear, distributed and time-varied. So we propose the multi-task real-time OS, which makes the system respond all external requests and control all real-time tasks in time, for embedded system designing.

A structure modeling method based on Tasks DFD is very suitable to such embedded system development. As a proving instance in this paper, we present the analysis and design procedure for an RTOS based IMS embedded system

development using our method. It is true that task-based DFD modeling is definitely an efficient solution for real-time embedded system development.

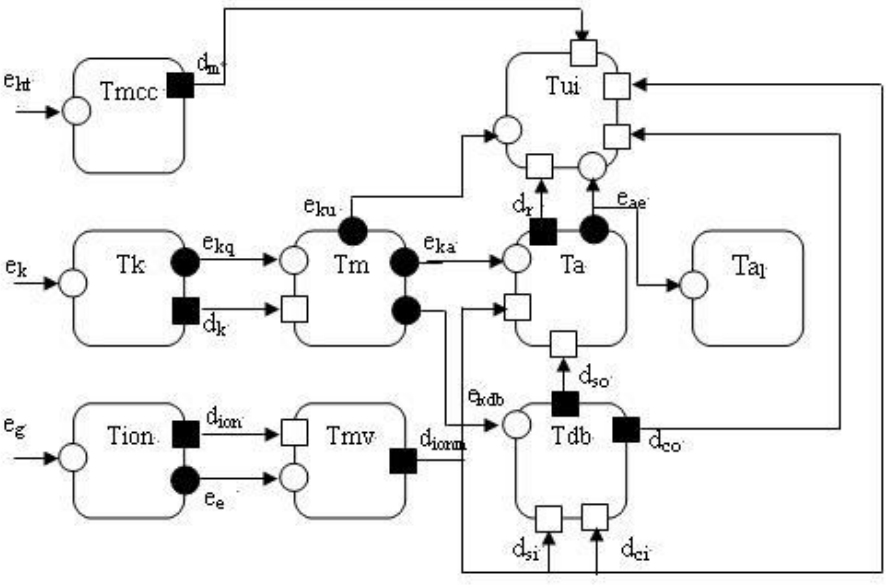


Fig.4. New Tasks DFD

References

1. Klein E., RTOS Design: How is Your Application Affected, Embedded Systems Conference Papers, ESC West 2000 San Francisco.
2. Xu Shuwu, Zheng Jian, Bi Zhiyi and Chen Yangzhou, Ion mobility spectrometry and its applications, Physics, 2003，32（8）539-542.
3. Wang Chunmin, Liu Zhenhua, and Guo Yunfei,Task Decomposition of Application Design in Real-time Operating System, Computer Engineering,2000，26（7）190-192.

Chaos-Model Based Framework for Embedded Software Development*

Huifeng Wu[1], Jing Ying[1], Xian Chen[2], Minghui Wu[3,4], and Changyun Li[1]

[1] College of Computer Science, Zhejiang University, HangZhou, 310027, China
{whf,lcy01}@zju.edu.cn,yingj@hangzhouit.gov.cn
[2] Department of Computing Science, University of Alberta,
Edmonton, T6G 2E8, Canada
xianchen@cs.ualberta.ca
[3] Department of Computing, The Hong Kong Polytechnic University,
Hong Kong, China
csmhwu@comp.polyu.edu.hk
[4] Department of Computer Science and Engineering,
Zhejiang University City College, Hangzhou, 310015, China

Abstract. The main issue of today's software development process is how to relate the schedule and coding of the software project. The chaos model of software development gave a theoretical description for it. This paper introduces the chaos model into the research of software architecture and brings forward a new software development framework CBFSD, which divides the process of software development into three levels: tasks level, codes level and components level. The construction of components level's architecture is the core of the whole framework. In this framework, we use the chaos strategy: "Resolve the most important issue first." The strategy makes it more appropriate to design and develop complicated software systems.

1 Introduction

After fifty years' development, today's software engineering confronts with more and more complicated software system and the organization of software development becomes more and more intricate. In the past, programming methodologies pay more attention to how to resolve technical problems than how to resolve users' problems or meet deadlines. Each aspect of software development is studied in isolation, instead of how to put them together. Lots of development models, such as waterfall model, spiral model and so on, only discuss issues of management level rather than how to write one line of code or fix one bug. These models have some shortages: because we must not only understand the flow of a project and how to write each line of code in the software development process, it is more important to understand how one line of code relates to the whole project and then establish a set of integrated software development system

* Supported by Fok Ying Tung Education Foundation(No.94030)

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 582–588, 2005.

[1,2]. All these facts lead to L.B.S. Raccoon brought forward the chaos model [3,4] based on some preceding software development models [1,5,6,7,2]. It gave a theoretic description for the relation of software architecture and coding.

In chaos model a complicated software project is made up of many fractals. The relationships among these fractals aren't clear. The development process of a project can be regarded as micro-process in bottom-level, complexity gap in middle-level and macro-process in top-level. In these three levels, the complexity gap [8] is most complicated and it bridges the goal of macro-process to the solution of micro-process. The size of complexity gap is determined by the scale of issue, tools and the architecture of project. To design and cope with complexity gap is developer's main tasks. "Resolve the most important issue first" is the chaos model's development strategy [9].

This paper introduces the idea of chaos model into software architecture and brings forward a chaos-model based framework for software development (CBFSD), and gives a detail description for components level's architecture, then establishes a development strategy and environment for it.

2 A Three-Layer Framework Based on Chaos Model

Generally it is a linear process to resolve a problem, which has a clear structure. Traditional software development models divide a development process into task definition, technical development and system integration linearly. The characteristic of it is that the relationships between subtasks are linear and clear. On the contrary, to cope with a complicated problem is more difficult, which turns out to be a chaotic process. The relationships between subtasks are not linear, since they have fuzzy and complicated relationships. So CBFSD divides the software development process into three levels:

1. **Tasks Level**. In this level the focus is the finished programs or products. The requirements will be divided into some specific tasks, and it takes the responsibility of organizing and scheduling these tasks. In this level, the tasks can be distinguished and dispatched strictly.
2. **Codes Level**. In this level programmers care about what technology can support their coding and the code's function. The function is based on the program tools, methods and a specific advanced language, and the codes realize the function.
3. **Components Level**. This level copes with chaos model's complexity gap. Complexity gap is "everything between the macro-process and the micro-process" [8]. In CBSFD, tasks level deals with the macro-process and codes level deals with the micro-process. Components level is in the middle of them and bridges the gap of tasks level and codes level. The components level has complicated structure, and it may include several subcomponents levels. The number of subcomponents levels is determined by the system's complexity. The components level embodies the components' plug-in architecture. The components level relies on the tasks level and codes level in the entire software development process. It is unstable and will have to change when the goal of tasks level or the technology of codes level changes.

The three-level structure of CBFSD describes the software's development process from different point of view. Adjacent levels influence each other very strongly, while distant levels influence each other very weakly.

Dealing with the complexity gap is the core task of chaos model. In CBFSD, components level covers the whole complexity gap, and the tasks switch from coping with the complexity gap to cope with the components, their relationships and alterations.

3 The Architecture of Components Level

In CBFSD, tasks level and codes level can be described by existent tools and approaches. For example, the tasks level can be defined by natural language or formal language, and the codes can be written by advanced languages. But it is more difficult to handle the components level, because developers must define and apply appropriate approaches to cope with the complexity gap in chaos model. So it is necessary to give a detail definition for the components level's architecture.

Component is the replaceable, reusable and independent unit, and it communicates with other components by a series of interface parameters. It can be expressed as a five-tuple:

(Task-Model, Property-Model, Sublevel-Set, Related-Set, Control-Interface)

Task-Model describes what tasks the component can accomplish. It can be a single task or a set of tasks.

Property-Model describes the components' properties, such as components' behavior and existence mode. It includes exterior properties and interior properties. The exterior properties determine how the component communicates with other components, and the interior properties determine the component's inherent properties.

Sublevel-Set describes the component's position in components level. The division of sublevels isn't strict, and a component may belong to a single sublevel or several sublevels. So we express component's position with the concept of set, which may includes only a single sublevel or several interrelated sublevels.

Related-Set is a set of interrelated components.

Control-Interface describes the component's control interface. It is the way how different components communicate with each other.

Component Plug-In describes the components' affiliation in components level. It can be described by a six-tuple:

(P-Component, S-Component-Set, Plug-In-Regulation, Sublevel-Set, Related-Set, Control-Interface)

P-Component describes the father component and it is exclusive.

S-Component-Set describes the set of son components.

Plug-In-Regulation describes the rules of plug-in. Components plug-in can be described as a component tree according to the rules. The rule's BNF can be described as follows:

$$< P - Component > ::= < S - Component - Set > \quad (1)$$

$$< S - Component - Set > ::= < S - Component > | < S - Component - Set > and < S - Component > \quad (2)$$

Sublevel-Set describes the position of component plug-in in components level. It is similar with the component that the component plug-in may belong to several sublevels.

Related-Set describes a set of interrelated component plug-ins.

Control-Interface describes the interface of component plug-in. It is the way of different component plug-ins communicate with each other.

Component alteration describes one alteration of components, the summation of all the alterations constitutes the evolvement process of components level, and it can be described as a three-tuple:

(Original-Component, Altered-Component, Alteration-Regulation)

Original-Component is the component before alteration.

Altered-Component is the component after alteration.

Alteration-Regulation is the rules of alteration.

Components alteration can take place in any phase of software development process. Obviously, the evolvement of components level must be simplified in order to master and summarize the rule of components' alteration-regulation for different kinds of components. Figure 1 shows a typical structure of components level.

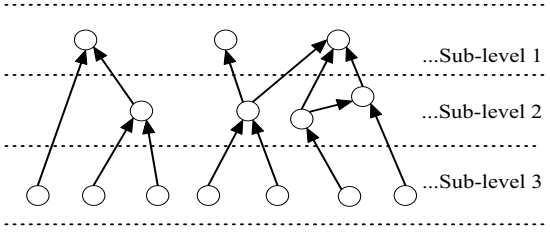


Fig. 1. A Typical Structure of Components Level

4 Development Strategy and Environment

CBFSD adopts chaos strategy, which is embodied in the components level mainly. The principle of chaos strategy is that the component has high priority will be developed first, and then the component with low priority.

Priority Level indicates the importance of component in software development process. The priority level is not rigid and quantitative, while it is a rank and general concept. It means that there may have some components have the same priority level. Which one of them will be developed first is determined by the state of development process and developers. Once a component has been

finished, a component alteration will take place, and then the components level's priority level phase will change accordingly.

Priority Level Phase (PLP) is made up of all the components' priority level in components level. Suppose there are k components, $C_i(i = 1, 2, ..., k)$, in components level, and their priority level are P_i, then the PLP is:

$$S_{PLP} = \bigcup_{i=1}^{k} P_i \tag{3}$$

The frequent change of PLP is the characteristic of a complexity system. It will help developer to master the process of software development to look into the PLP and simplify its structure.

Succeeding component is a component that will be developed after the given component is finished. The succeeding component of C_i is Cs_i.

Component Development Section is a series of components. Suppose $k_i \in N$, $C_{k_{i+1}}$=$C_{s_{k_i}}$, then a component development section is $C_{k_1}, C_{k_2}, \ldots, C_{k_n}$. And we call it as a development section of C_k. The development process of components level is made up of many interrelated development sections.

In order to establish the PLP, it is necessary to set the priority of every component. Priority is a very subtle concept and will change according to context. So the priority of every component should be reset at every moment when a component finished. Usually the component has high priority should be reusable, big, urgent, robust and have explicit requirement.

Reusable means whether there is a reusable component satisfying our requirement or the component will be reused by other modules in the software system.

Big means the component can make major progress for the entire project.

Urgent emphasizes timeliness and response to critical concerns and threats to hold up the project.

Robust means the component increasing of simplicity and flexibility of the entire project.

Explicit requirement means the component that has fine requirement specification and needs not to be modified any more.

If a component has all these properties at the same time, it should be developed first and has the highest priority. If a particular component lacks one or more these priorities, then its priority is lower than other components with more properties.

5 Application and Estimation

We have applied the CBFSD in some software development processes and realized a multi-agent software environment that embodies chaos strategy primarily.

In a multi-agent software system, the design process of every agent can be regarded as a linear process. The collaborative work of multi-agent can form a structure according to actual requirement. The design process is a chaotic

process. Figure 2 indicates the architecture of multi-agent software design based on CBFSD. Every ellipse in pane is the components level's design process of multi-agent software.

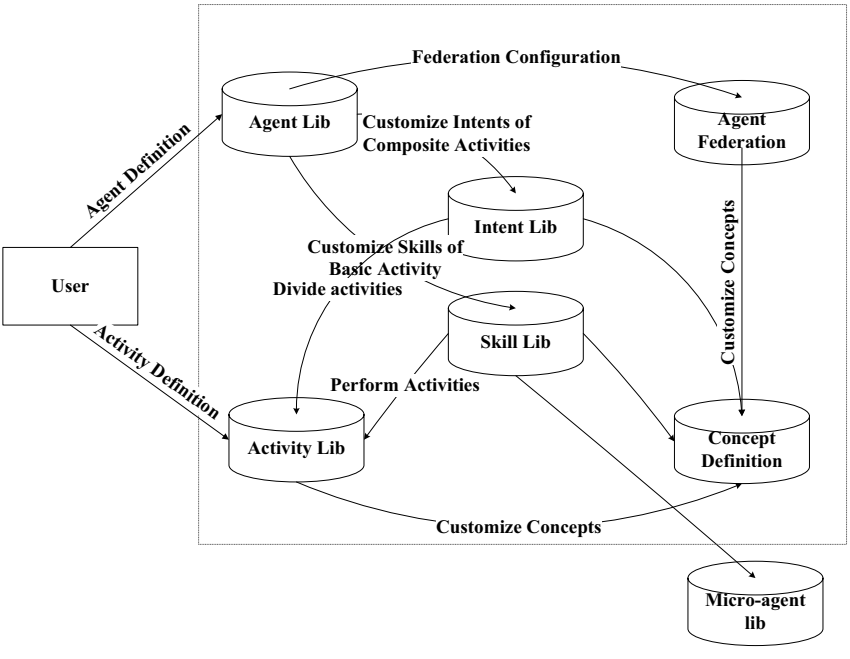


Fig. 2. Architecture of Multi-agent Software Design based on CBFSD

There are several ways afforded to developers to design and configure multi-agent software system, the detail is as follows:

1. Taking the activity of design and division as main route. The system's goal should be established first by developers. It means which activities should be finished and how the activities can be divided into subactivities. At the same time, the properties of these activities are configured. Then developers should determine which subactivities belong to every agent, and define the skills or intents of every agent to carry out the activities. At the same time, developers define and divide intent rules. These agents constitute a federation at last.
2. Taking the abilities of agent as main route. First make sure what skills and intents the agent holds. According to the skills and intents, ascertain which activities it can carry out. Then associate these skills and intents with the activities, afterwards, divide the activities into subactivities and configure their properties. These agents constitute a federation at last.
3. Taking the agent federation as main route. First, developers make sure the organization structure of agent, which means determining which agent fed-

erations the whole system is made up of; which members and familiars every agent holds and the agent's ability (which activities the agent can carry out, and which skills and intents the agent holds and what properties every skill and intent possesses).

The figure indicates that the ultimate goal of every design approach is the same, and only design mode and sequence are different from each other. But there is no formula for us to use. The developers can lower priority level and then use the above approaches alternatively according to actual requirement in the design process. With this pattern, we improve the interaction between the system and developers. Developers can improve the design with their ideas in the design process to make the design process more effective and flexible.

6 Conclusions

The application of chaos model in software engineering is a brand-new research field. The chaos model focuses on three important aspects, which are chaotic organization, complexity gap and chaotic strategy and environment. It is an effective method to deal with complicated and large-scale software system. In this article, we bring forward CBFSD based on chaos model, and establish an elementary foundation for the research in this field. We have done some work on it, which points out the direction for our subsequent research.

The further goal of CBFSD is to improve the development environment's usability and extensibility, and makes it suitable for the software development.

References

1. Berard., E.V.: Essays on Object-Oriented Software Engineering. Prentice Hall, New Jersey (1993)
2. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy., F.: Object-Oriented Modeling and Design. Prentice Hall, New Jersey (1991)
3. Raccoon., L.B.S.: The chaos model and the chaos life cycle. ACM SIGSOFT Software Engineering Notes **20** (1995) 55–66
4. Raccoon., L.B.S.: Fifty years of progress in software engineering. ACM SIGSOFT Software Engineering Notes **22** (1997) 88–103
5. Boehm., B.W.: A spiral model of software development and enhancement. IEEE Computer **21** (1988) 61–72
6. Boehm., B.W.: Using the win win spiral model: A case study. IEEE Computer **31** (1998) 33–44
7. Royce., W.W.: Managing the development of large software systems: Concepts and techniques. In: 1970 WESCON Technical Papers, Western Electronic Show and Convention, Los Angeles (1970) A/1–1–A/1–9
8. Raccoon., L.B.S.: The complexity gap. ACM SIGSOFT Software Engineering Notes **20** (1995) 37–44
9. Raccoon., L.B.S.: The chaos strategy. ACM SIGSOFT Software Engineering Notes **20** (1995) 40–47

Hierarchical Integration of Runtime Models*

Cheng Xie[1], Wenzhi Chen[1], Jiaoying Shi[1], Lü Ye[2]

[1] College of Computer Science, Zhejiang University, Hangzhou 310027, P.R.China.
arthurxie@vip.sina.com,
wzchen@cad.zju.edu.cn, jyshi@cad.zju.edu.cn
[2] Department of Computer Science and Electronics Engineering, ZheJiang University of Science and Technology, Hangzhou 310012, P.R.China.
yelue2004@yahoo.com.cn

Abstract. The complexity of embedded applications is growing rapidly. Mainstream software technology is facing serious challenges for leaving out non-functional aspects of embedded systems. To achieve this goal, we have defined a component-based modeling and assembly infrastructure, Pcanel, that supports hierarchical integration of concurrent, runtime models. A key principal in Pcanel is its netlist, namely component connection network. Pcanel advocates netlist as global view of a systemic design, where the basic building block is component. The functionality of embedded system is modeled as netlist. The communication among components is modeled as token flow. The distribution of functionality on netlist is transparent from the runtime models, which makes communication refinement easier. When applied formal models to components, the resulting runtime netlist maintains assurance of diversified non-functional aspects, such as timing and deadlock. The infrastructure advances the synergy between design-time models and runtime models.

1 Introduction

The complexity of embedded applications is growing rapidly. Mainstream technology of embedded software development is facing serious challenges. While reliability standards for embedded software remain very high, many new requirements of embedded software are desirable, such as rapid deployment and update, much more dynamic reconfiguration, low-power mobile computing, multimedia signal processing, etc. Prevailing abstractions of computational systems leave out these non-functional aspects of embedded systems, so that the methods used for general purpose software require considerable adaptation for embedded software. To achieve this goal, we have defined a component-based modeling and assembly infrastructure, Pcanel, which supports hierarchical integration of concurrent runtime models for component-

* This work was supported in part by the Hi-Tech Research and Development Program of China (863 Program) under Component-based Embedded Operating System and Developing Environment (No.2004AA1Z2050), and Embedded Software Platform for Ethernet Switch (No. 2003AA1Z2160); In part by the Science and Technology Program of Zhejiang province under Novel Distributed and Real-time Embedded Software Platform (No. 2004C21059).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 589-594, 2005.

based embedded system construction. Pcanel emphasizes modeling, not any more interface definition, functional customization and hardware management.

Pcanel structures the development process into two hierarchies. The top hierarchy represents the abstract design of a system in terms of functionality, leaving out specific implementation details, such as hardware configuration, middleware and fault tolerance. The bottom hierarchy refines the design by realizing non-functional aspects in a structural and systematic way. For example, abstract components can be mapped to operating system processes, connections between components are supported by distributed middleware, and specific implementations are selected from component repository for requirements of fault-tolerance and real-time. The essential design problems are solved by modeling and analysis before final implementation. During recursive refinements viz. hierarchical integration of runtime models, the non-functional aspects including time, concurrency, correctness, reactivity, and heterogeneity are integrated into the design, until it can be finally synthesized to implementation. In this infrastructure, atomic component is built from computation blocks, and complex component is recursively built from composition of fine-grained components. Each component communicates with others under particular model of computation. The behavior of component is modeled as a set of transitions. A composite component is an encapsulated framework that consists of a runtime model and a graph of connected components. Through hierarchical integration of runtime models, Pcanel has great capability in separating systemic design from behavior specification, and capturing the requirements and constraints of the system.

2 Hierarchical Integration

Pcanel advocates netlist, namely component connection network, as global view of systemic design, where the basic building block is component. A component is a computational entity having a set of transitions. Components have interfaces defined by a collection of abstract input/output ports. Ports are shared states that allow components to communicate with each other via tokens. A set of connected ports represent a channel, through which a model drives the flow of tokens. A framework consists of a model and a graph of components, allowing for the observation and manipulation of the runtime states and behaviors internal of components.

Furthermore, to facilitate modularity, a framework itself, together with the components under its control, can be treated as a single component at a higher level of hierarchy, which means that the framework can be encapsulated to a composite component. Thus, a complete system configuration is a set of hierarchical compositions of models and components. Figure 1 shows two hierarchies of composition. The framework B is encapsulated into a composite component b by introducing more states. The transitions of {(3,4), (4,5), (5,6)} in B is abstracted as an atomic transition (1,2) in A. When applied formal models to components, the resulting composite component maintains assurance of diversified non-functional aspects, such as timing and deadlock.

Models are independent of implementation of components. Based on Pcanel, an embedded system can be built rapidly by reusing existed components and customizing netlist. Pcanel implements several models of computation for complex embedded

system design, including continuous time (CT), discrete event (DE), synchronous dataflow (SDF), communicating sequential processes (CSP), Priority-driven multitasking (PDM), and finite state machine (FSM), etc.

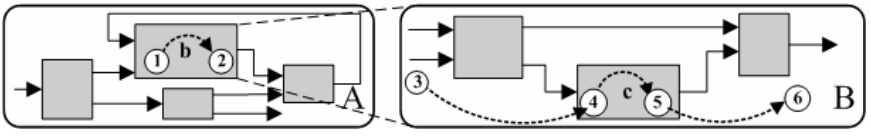


Fig. 1. Hierarchical Composition

The integration of hybrid runtime models of a two wheeled vehicle is shown in Figure 2. The vehicle, called Cyveh [11], is principally a self balancing machine with fully automated driving capabilities, whose wheels share a common axis. To implement the balance system of Cyveh, the control software is composed by hybrid runtime models including CT, DE, FSM, Modal and SDF.

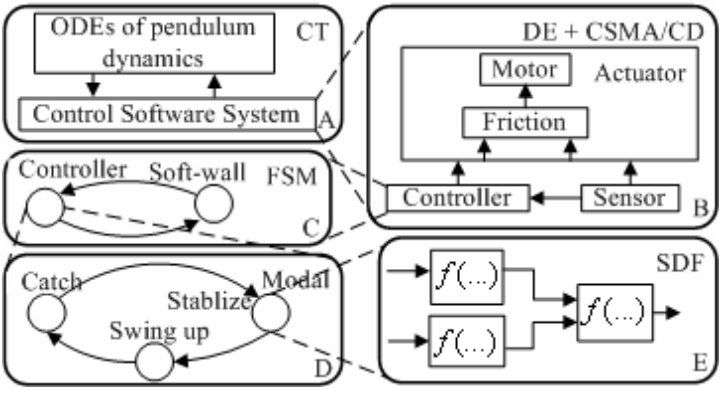


Fig. 2. Integration of runtime models in Cyveh

The top framework A is driven by CT model and contains two components, one is a set of differential equations modeling the physical pendulum dynamics, and the other is the control software. The framework B implements the control laws under DE model, which contains the actuator, the sensor and the controller. For heterogeneous road surface, a component serving friction compensation can be dynamically loaded into the actuator [4]. The framework C of the controller is driven by FSM model of two states. Any time the Cyveh enters a protected area, e.g. too close to other vehicles, the controller switches to the soft-wall state. A force is applied in the opposite direction to avoid collision. The framework D implements the normal operation within a modal model of three modes. Initially, the swing-up mode brings Cyveh from lean parking to upright position by energy control [5]. Once it is sufficiently close to upright position, the controller switches to the catch mode that slows down the pendulum body rotation before entering the third mode, stabilize. At last, the Cyveh keeps balance when the stabilize component is running. The framework E driven by SDF model implements a control algorithm that maintains the natural equilibrium point of the Cyveh system.

3 Component Netlist

Typical distributed embedded system consists of a network of nodes connected via bus or network. As the platform architecture shown in Figure 3, each node consists of the processor, an operating system, a dedicated communication layer, and one or more application transactions. The complete software can not independently from the hardware. The execution of software depends on the underlying processor architecture, memory mapping, bus of SoCs, or device registers. For reuse of components, hardware platform profile is included in the description of node.

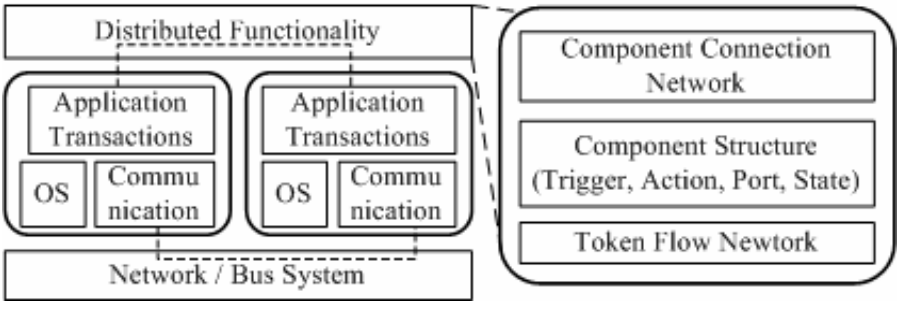


Fig. 3. Platform architecture

The behavior of a component is modeled as component structure defined in [11]. The behavior of a system is modeled as component netlist of hierarchical runtime models. The communication in netlist is modeled as token flow being carried out on token basis. The token flows are scheduled under models. In a hybrid system, hierarchical heterogeneous models cooperatively direct the token flows. A global token flow network can be constructed from the component netlist for analysis and verification of concurrent and real-time aspects in a hybrid system.

The distributed mapping from functionality to nodes is vertical to the netlist. The communication between components on spatially separated nodes is wrapped by the communication layer. The communication layer defines how software can be integrated with given components. The integrated system model may span hybrid bus systems, such as Controller Area Network and Local Interconnect Network. Since each reusable component is implemented with a set of transitions that uniquely define its functionality without side effects, components can be refined into the netlist based on their design specifications.

Models of computation are independent of implementation of components. Thus, Reusable components in integrated software are organized hierarchically to support integration with different models. A complete system configuration, i.e. the component connection network, is actually the synthesis result of hierarchical composition of reusable components. The netlist consists with the models of computation, thus allows for the observation and manipulation of the runtime states and behaviors internal of components. Such a netlist supports hierarchical composition, which is able to keep the global overview of the system.

4 Synergy Between Design and Runtime

Several approaches to the composition of software from components have been proposed in the literature [9, 7, 6]. An important contribution to this topic stems from the field of software architecture systems. Architecture systems introduce the notion of components, ports, and connectors as first class representations. In [8] a component model is used for embedded software in consumer electronic devices. In [1] a framework for dynamically reconfigurable real-time software is presented. It is based on the concept of so called Port Based Objects. However, most of the approaches proposed in the literature do not take into account the heterogeneous properties of software for hybrid systems.

Systematically integrating heterogeneous components is crucial to design large-scale distributed real-time systems. Many active research projects address this issue and influence our design. For example, [3] proposes a globally asynchronous and locally synchronous (GALS) architecture, that asynchronous message communication is used to maintain the synchronous semantics of execution of components and their composition. [10] integrates multi-rate time-triggered architecture with finite state machines. But most of these projects only integrate two models and assume a fixed containment relation between them. These architectures lack formal runtime models for composite components. Our infrastructure enables hierarchical heterogeneous compositions along well-defined models that are semantically separate from one another. In addition, unlike these approaches, our work has a strong emphasis on runtime systems.

It is important to advance the synergy between heterogeneous design environments and runtime systems. Design-time environments emphasize the understandability of models, syntax and semantics checking (like type systems), and component polymorphism. Ptolemy II [2] supports the modeling, simulation, and design of concurrent, real-time, embedded systems. It incorporates a number of models of computation (such as synchronous/reactive system, communicating sequential processes, finite state machine, continuous time, etc.) with semantics that allow domains to interoperate. On the other hand, runtime systems emphasize physical interface, performance, and footprint. Not all design-time models are suitable for direct implementation on runtime systems. Except for models that only are useful for modeling physical environment, certain models transformed to embedded software may be nondeterministic, inefficient and deadlock.

The rough approach of integrating heterogeneous runtime models is to implement them indirectly by a grand unified model. For example, it is possible to emulate most models on a time-synced priority-driven model provided in traditional RTOS. The methodology attempts to build a flat layer of abstraction fit for all applications. However, grand unified models are usually difficult of analysis and synthesis. In addition, an application usually does not need all the features provided by the grand unified model. Mixed features degrade performance and take overstaffed footprint.

Code generation approach adopted in Ptolemy II project is a migration path from certain design-time models to runtime models. The recent Ptolemy II release includes a limited prototype of code generation facility that will generate a stand-alone program of java class files for non-hierarchical SDF models. However, the code generation process is a very restricted solution for the wide heterogeneity and

irregularity of embedded systems, and the result program is not portable for variant operating systems.

We argue that a hierarchical runtime infrastructure natively supporting executable models will greatly help code generation and improve the quality of final software. A runtime system can utilize hardware support (such as SMP) and communication systems (such as CAN) to provide high responsible frameworks to applications. In addition, there are certain assumptions, like resource reservation and timing predictability, can only be achieved by OS-level runtime systems, but not easily by stand-along programs.

5 Conclusion

Noticing a wide variety of design-time models for distributed, real-time, embedded systems, this paper motivates a component-based modeling and assembly infrastructure, Pcanel, to integrate heterogeneous executable models and support composition of components. Pcanel proposes a runtime infrastructure for constructing responsible systems through runtime models integration. A key principal in the infrastructure is its component netlist, which makes the runtime system responsible for the distributed functionality. The novel design of Pcanel advances the synergy between heterogeneous design environments and runtime systems.

References

1. D.B.Stewart, R.A.Volpe, and P.K.Khosla. Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects. IEEE Transactions on Software Engineering, 1997.
2. E. A. Lee. Overview of the Ptolemy Project. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, Berkeley, July 2, 2003.
3. E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: A programming model for event-driven embedded systems. Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing, 2003.
4. E. Garcia, P. Gonzalez, and C Canudas de Wit. Velocity dependence in the cyclic friction arising with gears. International Journal of Robotics Research, 21(9):761–771, 2002.
5. K. Astrom and K. Furuta. Swinging up a pendulum by energy control. IFAC 13th World Congress, San Francisco, California, 1996.
6. M. Shaw and D. Garlan. Software Architecture - Perspectives on an Emerging Discipline. Prentice Hall, 1996.
7. Paul C. Clements. A survey of architecture description languages. International Workshop on Software Specification and Design, 1996.
8. Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. IEEE Computer, 2000.
9. Robert Allen and David Garlan. A formal basis for architectural connection. ACM Transactions on Software Engineering and Methodology, 6(3):213– 49, July 1997.
10. Thomas A. Henzinger, Christoph M. Kirsch, Marco A. Sanvido, and Wolfgang Pree. From control models to real-time code using Giotto. IEEE Control Systems Magazine, 2003.
11. Cheng Xie, Wenzhi Chen, Jiaoying Shi. Pcanel: A Model Driven Component Framework. IEEE Conference on Systems, Man and Cybernetics, 2004.

Object-Oriented Software Loading and Upgrading Techniques for Embedded and Distributed System

Bogusław Cyganek

AGH - University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
cyganek@uci.agh.edu.pl

Abstract. The paper addresses the problem of loading and upgrading mechanisms for embedded and distributed systems. Such mechanisms for most of the large operating systems have been developed over the recent years. However, custom embedded systems usually lack a proper solution, mostly due to their hardware varieties. In this paper the object-oriented design is presented that allows for reliable software loading and upgrading for many architectures of embedded systems regardless of their complexity and computational power. The proposed solution was implemented and tested on the real platform and showed great robustness. The paper can be of interest for designers of embedded and distributed computer systems.

1 Introduction

The paper addresses the problem of software loading and upgrading of embedded and distributed microprocessor platforms. Such systems have been developed for the well known operating systems [7][8] and most of the commercially available embedded platforms [11]. Specifics of loading and upgrading of distributed systems with further references can be found in [1][2]. However, custom designs usually lack a ready solution to this problem. In this paper we propose our solution that can pose a common starting point for other projects as well. The presented system is defined in object-oriented terms and therefore it is easily adaptable to custom systems regardless of computational power of their building modules. The implementation was tested in a real embedded system designed for the specific customer hardware.

2 Design of the Loading and Upgrading System

The proposed loading and upgrading system follows the server and client(s) pattern [4][5]. The overall view of this system is depicted in Fig. 1. The essential thing is a communication channel present among participants of this architecture. Fig. 2 presents internal structure of the server and clients modules. They usually consist of many building blocks or subsystems (such as hardware modules HM or memory

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 595-600, 2005.

banks, etc.). From the loader/upgrader system point of view the Flash memory and communications facilities are essential, however.

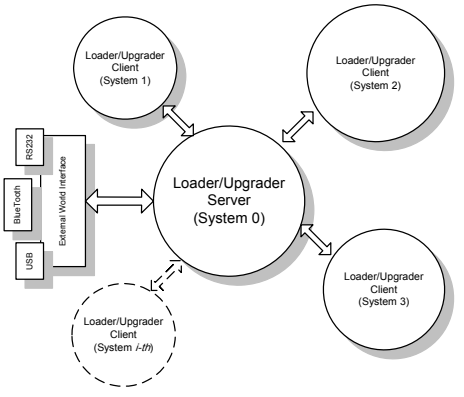


Fig. 1. Server/Client architecture of the upgrader/loader system

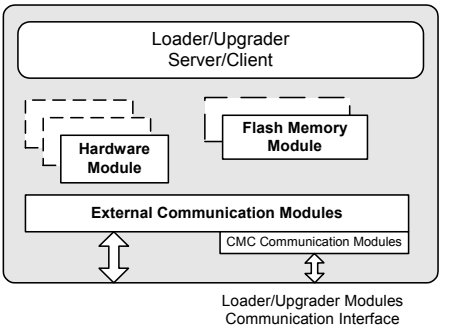


Fig. 2. Structure of the server and client modules

The following statements describe our concept of the loader/upgrader system:

1. The loader/upgrader server (LUS) resides only on one (host) system (System 0).
2. The loader/upgrader clients (LUC) are software components present on each of the participating systems (i.e. Systems 1 to n in Fig. 1).
3. There is a communication channel for control and data exchange among the participants.

There are no additional constraints on the structure of the underlying microprocessor systems – it can be even a distributed configuration. The communication channel (in Fig. 1 and Fig. 2) can be any data transmission channel with a protocol. In our experiments the Control Messaging Channel (CMC) was used [3]. It is also assumed that there are means of external communication for data exchange. In Fig. 1 this is denoted as optional connections of the system with LUS on board with an external environment (e.g. PCMCIA, USB, Ethernet, Internet, RS232, etc.) [6][10].

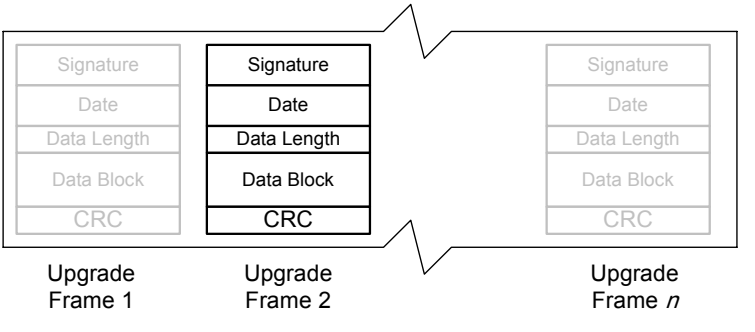


Fig. 3. Format of the upgrading data modules

Fig. 3 depicts the proposed format of input data. The "Signature" field uniquely identifies a given data block. This ID value governs choice of a proper software module for subsequent load/upgrade action. The field "Date" contains a 32 bit encoded date of a given data block. This information can be used by a right upgrade guide entry module (an upgrade-cartridge) to judge whether existing data should be obliterated by the new one. "Data Length" contains a number of data bytes, i.e. length of the "Data" space. "Data" contains new data. The "CRC" field is a two- byte field containing the CRC value computed from the whole frame except this field. In this modified version the CRC-16 polynomial ($x^{16}+x^{15}+x^2+1$) was used.

The semantic specification of the proposed L/U system is as follows:

1. The upgrade file is provided to the LUS by an external data connection.
2. The LUS analysis a upgrade frame and tries to find a proper upgrade cartridge (previously registered to the system). The choice is made based on the signature conveyed by each upgrade frame. If such a module is found on the server's system then an action is delegated to it. Otherwise, the upgrade frame is transferred to the chain of other sub-systems by means of the control messaging channel (see Fig. 1).
3. Once a valid upgrade frame is detected on other system then again the proper upgrade cartridge is searched that complies with the signature field of this upgrade frame. If found, then the upgrade process is delegated to that module.

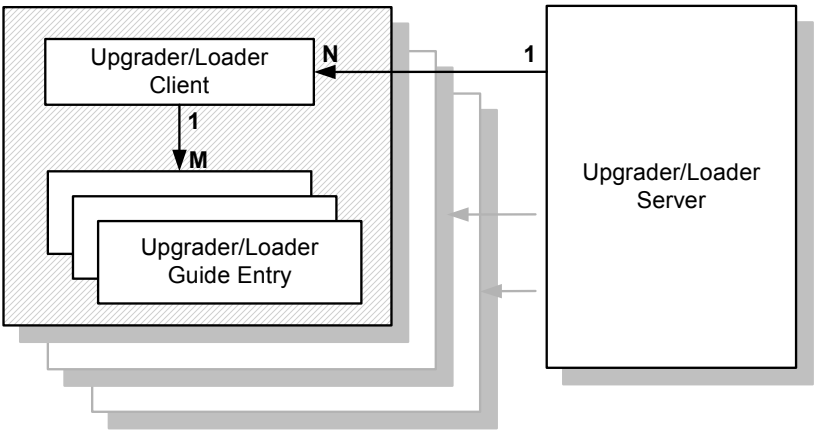


Fig. 4. The mutual relationship among upgrading objects (object semantics)

Fig. 4 presents three kinds of objects participating in the described load/upgrade scenario, as well as their mutual relations. These are:

1. The upgrade server object – responsible for communication and processing of the upgrade file, its disassembly onto upgrade frames and resending to the available clients, registered into this server. There is usually only one server object.
2. The upgrade client object – accountable for reception of an upgrade frame from the server, search for an upgrade cartridge (i.e. the upgrade guide entry) and action delegation to this cartridge (if found). There can be many clients in the system.
3. The upgrade guide entry object – a specific load/upgrade processing module that specializes in load/upgrade for a specific frame and system. There can be many

such objects registered to a client. It is also possible to register a single upgrade guide entry to many clients.

3 Object-Oriented Implementation

Realization of the proposed upgrader/loader system is presented in object-oriented terms [9]. However, implementation can be *specific* to each of the sub-systems separately. It has only to comply with the defined functionality and object semantics.

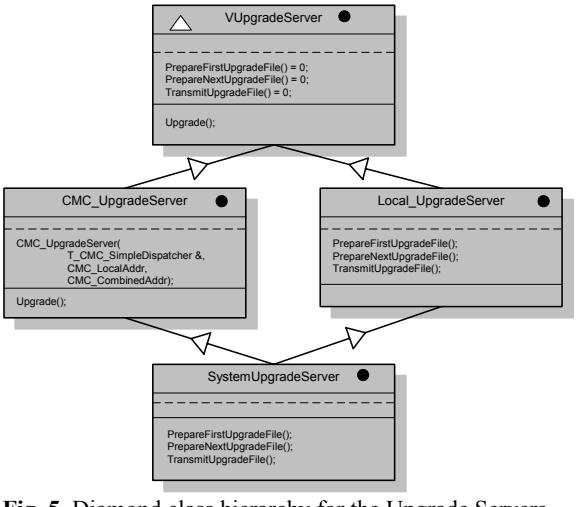


Fig. 5. Diamond class hierarchy for the Upgrade Servers

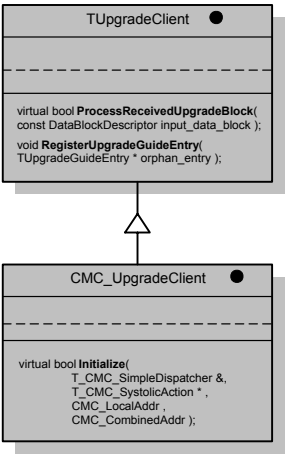


Fig. 6. The Clients hierarchy

3.1 The Loader-Upgrader Servers (LUS)

The loader/upgrade servers perform two actions on account of the system:
1. Registration of upgrade client object(s).
2. Extraction of a consecutive upgrade frame from an upgrade file (Fig. 3).
3. Distribution of frames among registered client(s).

The upgrade server class hierarchy is presented in Fig. 5. The main server actions are defined in the virtual base class. The virtual iterators *PrepareFirstUpgradeFile()* and *PrepareNextUpgradeFile()* are responsible for disassembly of the input upgrade file. Both are implemented by derived classes. *TransmitUpgradeFile()* transmits an upgrade frame to a registered client (if such is found) based on the "Signature" field. The *CMC_UpgradeServer* prepares a connection via the communication channel (it can be any data link; in our realization it was CMC [1]). Its implementation of the *TransmitUpgradeFile()* method transmits an upgrade frame to the distributed system(s) by established data connection.

The *Local_UpgradeServer* is a specialized server accountable for communication with clients local to that server, i.e. ones resident in the same sub-system (such as

System 0 in Fig. 1). It creates its own upgrade client to which it directly delegates all upgrade frames that concern directly its platform. This server specializes the iteration (virtual) methods *PrepareFirstUpgradeFile()* and *PrepareFirstUpgradeFile()*, since the iteration is done entirely on its own platform.

The doubly inherited *SystemUpgradeServer* embraces functionality of its two bases. The only specialization is done in the *TransmitUpgradeFile()* method. A given upgrade frame from an upgrade file is delegated at first to the native servers' system. If it is accepted for an upgrade process then the action is done. Otherwise a frame is sent to other sub-systems. There an upgrade action is then attempted. Operation results are sent back by means of system communication facilities.

3.2 The Loader-Upgrader Clients (LUC)

Fig. 6 depicts hierarchy of the loader/upgrade clients, which are responsible for:
1. Registration of upgrade guide entries (i.e. specific upgrade cartridges).
2. Reception and consistency check of upgrade frames.
3. Distribution of upgrade frames among registered upgrade guide entry objects.

3.3 The Upgrade Guide Entry

The upgrade guide objects (Fig. 7) realize specific upgrade actions for different modules of the systems. *TUpgradeGuideEntry* is a pure virtual base. Other classes are specialized versions for specific data loading/upgrading (e.g. FPGA bits loading).

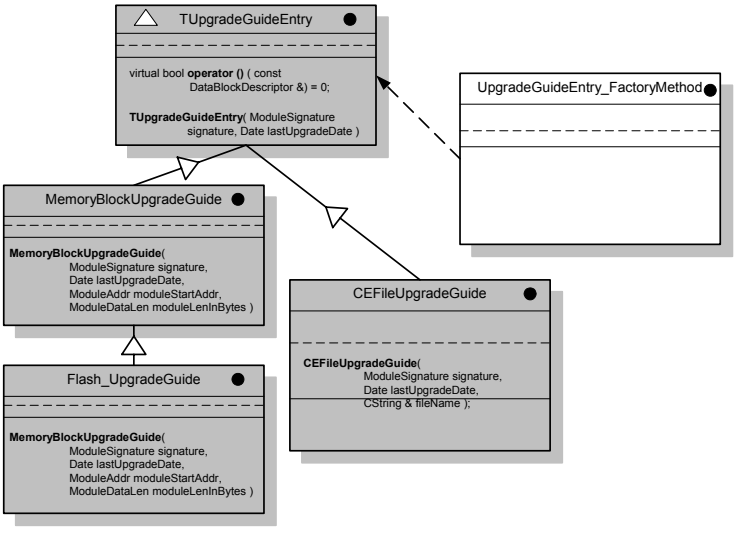


Fig. 7. Class hierarchy for the Upgrade Guide Entries

The specialized *MemoryBlockUpgradeGuide* class is responsible for updates on operational memory. Its specialized derivative, the *Flash_UpgradeGuide* class, does the same action but with respect to the Flash memory on sub-systems. The

CEFileUpgradeGuide is intended to upgrade files on the platform (in our case it was WinCE based). Additional upgrade guides can be freely added (by means of registration to the server) without any change to the existing configuration.

4 Software Testing and Experimental Results

The presented system was implemented in C++ and assembly. The experimental embedded platform consisted of the following sub-systems: the main board with the ARM processor running WinCE® 3.11, three slave-measurement boards with PowerPC 823 processors and two sensor boards endowed with the 80552 processors. The CMC protocol was implemented to provide the communication and data channels [1]. For more critical systems the reliability and security should be also considered and thoroughly tested. Especially important is a reliable operation of the LUS and LUC modules since any failure here can cause lost of control over the whole system.

5 Conclusions

This paper presents a proposition of the loading and upgrading system for embedded and distributed computer platforms. Description of the main three building blocks: servers (LUS), clients (LUC), and guide entries, as well as detailed descriptions have been also provided. The practical realization showed great robustness of the presented loader/upgrader system. Thanks to the object-oriented notation, the techniques can be easily implemented on any embedded platform that is able to communicate with other modules. Presented exemplary realization showed its great robustness in practice. It was also tested in terms of no-failure-operation-time. In future the presented loading and upgrading system can be easily changed or extended to fit custom designs.

References

1. Ajmani, S.: A Review of Software Upgrade Techniques for Distributed Systems. Technical Report, MIT Computer Science and Artificial Intelligence Laboratory (2004)
2. Ajmani, S.: Automatic Software Upgrades for Distributed Systems. PhD, MIT (2004)
3. Cyganek, B., Borgosz, J.: Control Messaging Channel for Distributed Computer Systems, Proceedings of the ICCSA 2004, Assisi, Italy, Springer LNCS 3046 (2004) 261 - 270
4. Douglass, B.P.: Doing Hard Time. Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. Addison-Wesley (1999)
5. Gamma, E., E., Helm, R., Johnson, R.: Design Patterns. Addison-Wesley (1995)
6. Halsal, F.: Data Communications, Addison-Wesley (1995)
7. Microsoft: Managing automatic updating and download technologies in Windows XP. http://www.microsoft.com/WindowsXP (2004)
8. Red Hat up2date. http://www.redhat.com (2004)
9. Taligent Inc.: Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++. Addison-Wesley (1994)
10. USB Org.: Universal Serial Bus Revision 2.0 specification. www.usb.org (2000)
11. Yaghmour, K.: Building Embedded Linux Systems. O'Reilly (2003)

Preserving Consistency in Distributed Embedded Collaborative Editing Systems*

Bo Jiang[1,2], Jiajun Bu[1], and Chun Chen[1]

[1]College of Computer Science,
Zhejiang University, Hangzhou, P.R.China
{nancybjiang, bjj, chenc}@zju.edu.cn
[2]College of Computer and Information Engineering,
Zhejiang Gongshang University, Hangzhou, P.R.China

Abstract. The increasing quest for mobility bolster the integration of technology of computer supported cooperative work and mobile computing. In cooperative editing systems that integrated with embedded devices, inconsistency is one of the major problems. In this paper, locking scheme and the corresponding algorithm of locking conflict solution are proposed to maintain consistency of the distributed shared documents and enhance the efficiency of collaborative users. The scheme and algorithm are realized in CoEdit – a prototype system of collaborative editing system.

1 Introduction

Computer-based groupware systems assist groups of users working simultaneously on common tasks by providing an interface and sets of collaborative tools for a shared environment. One of commonly used groupware systems is the real-time collaborative editing system (CES) [1], [2], [3], [4], which allows people to view and edit the same document at the same time from geographically dispersed sites connected by networks. CES is a very useful facility in Computer-Supported Cooperative Work (CSCW) application, such as collaborative authoring, collaborative design, and electronic meeting.

Nowadays, the issues of "ubiquitous computing" and "mobility" have received much attention in the CSCW literature. Mobile networks and embedded facilities make collaborative work more conveniently and can greatly enhance the efficiency. However, so far few researchers have explored the topic of collaborative editor with embedded devices in mobile environment.

This paper presents a research project with the object to design a cooperative editing system integrated with embedded devices that can maintain consistency of the shared document in mobile work. Since cooperative editing systems deployed on

* This paper is supported by National High Technology Research and Development Program of China (863 Program, No. 2004AA1Z2390) and Key Technologies R&D Program of Zhejiang Province (No. 2005C23047 & No. 2004C11052)

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 601-606, 2005.

embedded devices are heavily constrained by low computing capability, limited storage and unreliable network, the prototype system - CoEdit achieves collaborative editing on such devices by means of web services [5], [6], [7], [8]. In a collaborative editing environment, consistency of shared documents may be violated. Therefore, maintaining consistency is one of the core issues in the design of this type of systems and will be the focus of this paper.

The structure of this paper is as follows: Section 2 outlines the overview of the architecture of the cooperative editing system that is integrated with embedded devices. Section 3 analyzes the locking scheme applied in CoEdit. Section 4 proposes locking conflict resolution in the system. Lastly, Section 5 presents conclusions of the paper.

2 Overview of CoEdit

CoEdit is a prototype system that enables users to edit document with stationary computers or mobile embedded devices. Documents become the primary part in the shared workspace of cooperative editors. Artifacts are any objects consumed or produced in a document. The artifacts defined in this system can be presented in a hierarchical structure. In CoEdit document can be defined as follows:

document ::= attribute | <section | section | …> | operation
section ::= attribute | < line | line | …> | operation
line ::= attribute | < word | word | …> | operation
word:=attribute | operation

The main characteristics of cooperative editing systems rely in the following three aspects: (1) Real-time – system should respond to local site user actions as quick as possible and latency for reflecting remote user's actions is only determined by external communication latency. (2) Distributed – cooperating users may reside on different sites with diverse devices connected by different communication networks. (3) Unconstrained – users are able to edit any part of the document at any time freely.

In the replicated architecture, documents in the shared workspace are replicated at the local storage of each participating site. For the traits of good responsiveness and unconstrained collaboration of the replicated architecture, it is always adopted [9], [10]. However, there will always be limited storage space and computational power on embedded devices. Adopting fully replicated architecture in cooperative editing systems integrated with embedded devices will not be suitable any more. In CoEdit, we adopt the partial replicated architecture as shown in Figure 1.

CoEdit deploys the system on desktop computers as well as embedded devices, such as PDA and mobile phone. In order to obtain real-time and unconstrained characteristics of CES, shared documents and collaborative tools that enable cooperating work are replicated on PCs and server. However, it is not reasonable and possible for embedded devices to keep the whole document for its limited storage resources. Therefore, only fractions that are mostly related to the users editing tasks are remained on mobile embedded sites. And the means of accessing CoEdit resources stored in the server by embedded devices is Web Services for the small storage space that mobile devices have.

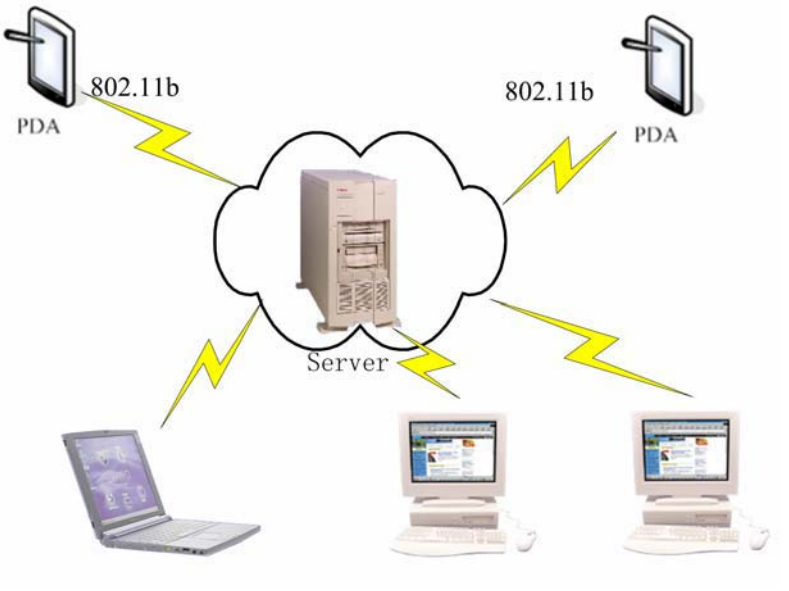


Fig. 1. Architecture of CoEdit

3 Consistency Preservation In CoEdit

In cooperative editing system, to preserve consistency of replicated documents is one of the major challenges in designing and implementing real-time cooperative editing systems. Consistency violation problem has been deeply researched and lots of schemes and algorithms have been presented in former systems [4]. Therefore, we only focus on the locking scheme which enables embedded sites to cooperate with each other.

According to the hierarchical structure of documents in CoEdit, locks can be set at different levels of granularity (e.g. document, section, line, etc.). In CoEdit both pessimistic locking and optimistic locking are adopted, also users can issue a locking operation to obtain the exclusive locking at random granularity level or lock set by system dynamically. Users can issue locking operations in certain granularity to avoid intrusion from other sites. The pessimistic locking can be simply expressed as follows:

PL:= (R_Id/R_Range)

R_Id represents the identification of the region, while R_Range denotes the range of locking region.

Besides pessimistic locking, optimistic locking is also adopted in CoEdit. A novel optimistic locking scheme that is proposed in this paper is dynamic locking. With dynamic optimistic locking, lock can be classified into two categories: shared or exclusive. Users may freely edit any unlocked region without issuing a locking command on it. While at the same time an implicit shared lock is placed on the region that the user is editing. The category and granularity of the lock will be changed dynami-

cally during the collaborative editing session. When one user edits a section, he will automatically gain a shared lock and the granularity of the lock will be sectioned. However, the section lock may be changed into fine granularity exclusive lock if multiple users are editing the same section. For example, as shown in Figure two, while two users cooperatively edit the same paragraph in different lines, they own exclusive locks on line 2 and line 4 respectively.

Computer-Supported Cooperative Work (CSCW) systems are computer-based systems that support groups of people engaged in a common task and provide an interface to a shared environment. Real-time collaborative editing systems (CES) are *very useful groupware tools in the rapidly expanding areas of CSCW systems*

Fig. 2. An example of dynamic locking

4 Locking Conflict Resolution

In a collaborative editing session, locking operations may arrive concurrently and lead to conflict. For instance, suppose two users edit the same sentence or two explicit pessimistic locking operations are issued but the locking regions are overlapped. Locking conflict problems may occur in these cases. Therefore, conflict should be detected and corresponding problems should be solved at the time when concurrent locking operations are issued no matter whether the locking operation is explicit or implicit.

There exist two classes of locking operation relation: one is conflicting and the other is compatible. The definition of conflicting relation and compatible relation are as follows:

Definition 1: Conflicting relation "⊗"

Given two locking operations LO_1 and LO_2 generated from site i and j, they are conflict with each other, expressed as $LO_1 \otimes LO_2$, iif (1) LO_1 and LO_2 are concurrent; (2) the regions related to LO_1 and LO_2 are overlapped.

Given two locking operations LO_1 and LO_2 generated from site i and j each, they are conflict with each other, expressed as $LO_1 \otimes LO_2$, iif (1) LO_1 and LO_2 are concurrent; (2) the regions related to LO_1 and LO_2 are overlapped.

In contrast, if two locking operations are not conflicting, then they are compatible, as defined below.

Definition 2: Compatible relation "©"

Given two operations LO_1 and LO_2 generated from site i and j each, if and only if LO_1 and LO_2 are not conflicting, they are compatible, expressed as $LO_1 © LO_2$.

Moreover, a group of operations may have rather arbitrary and complex conflict relationships among them in a highly concurrent real-time collaborative editing environment. Consider the following scenario, $LO_1 \otimes LO_2$, $LO_2 \otimes LO_3$, but $LO_2 © LO_3$. In order to solve such kinds of problem, a multi-version approach is proposed.

Given a group of n locking operations, LO_1, LO_2, ⋯, LO_n, targeting the common section, their locking conflict relationship can be fully and uniquely expressed by a n × n matrix, in which element M[i, j], 1⩽i, j⩽n is filled with "⊗" and "©". For example, 4×4 matrix for four operations is shown in Table 1.

Table 1. Relation between four locking operations

Relation	LO_1	LO_2	LO_3	LO_4
LO_1		©	⊗	©
LO_2	©		⊗	⊗
LO_3	⊗	⊗		©
LO_4	©	⊗	©	

Algorithm 1: Given a matrix of a group of N concurrent LOs. Locking Compatible Group Sets (LCGS) can be expressed as follows:

$LCGS = \{LCG_1, LCG_2, \ldots, LCG_n\}$

All operations in a single LCG_k are mutually compatible.

1. LCGS={};
2. For $1 \le i \le N$, and $i < j \le N$
   If M[i, j-1]= ©,
   Then $LCGS=LCGS+\{\{ LO_i , LO_j \}\}$
3. For $1 \le i \le N$
   If LO_i is not in LCG_k, $1 \le k \le |LCGS|$
   Then $LCGS=LCGS \cup \{\{LO_i\}\}$.

By using the above algorithm, the LCGS is formed. Through users' negotiation, only one LCG is selected to be applied if there are more than one LCG in the LCGS.

5 Conclusion

With the current proliferation of embedded technologies and rapid move towards use of mobile technologies, cooperative editing activities are expanded to embedded devices. In this paper, we present a cooperative editing system that can be deployed on both traditional stationary PCs and embedded devices. Embedded sites are able to access the remote server by means of web services to participate in the collaborative editing process. To tackle the problem that multiple embedded users may edit the same region in the shared document and maintain consistency, we present the locking scheme that can either be pessimistic or optimistic. And the algorithm to solve the locking conflict problem is proposed in the paper. The algorithm has been tested in CoEdit prototype system and the schemes make collaborative edit more efficiently.

References

1. Bo Jiang, Chun Chen, Jiajun Bu: CoDesign-A collaborative pattern design system based on agent. Proceedings of the Sixth International Conference on Computer Supported Cooperative Work in Design, Canada, (2001) 319-323
2. R.E. Newman-Wolfe et al.: MACE: A Fine Grained Concurrent Editor. Proceedings of ACM COCS Conf. Organizational Computing Systems. 240-254
3. M. Ressel, D. Nitsche-Ruhland, and R. Gunzenbauser: An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors. Proceedings of ACM Conf. Computer Supported Cooperative Work. (1996) 288-297
4. C. Sun and D. Chen: Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems. ACM Transactions on Computer-Human Interaction. (2002) 1-41
5. S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, andR. Weber: Active XML: Peer-to-Peer Data and Web ServicesIntegration (demo). Proc. of VLDB, (2002)
6. Curbera, F., Mukhi, N., Weerawarana, S.: On theEmergence of a Web Services Component Model. InProc. of the WCOP 2001 workshop at ECOOP 2001(Budapest, Hungary, June 2001)
7. Tosic, V., Esfandiari, B., Pagurek, B., Patel, K.: On Requirements for Ontologies in Management of Web Services. Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web - WES (at CaiSE'02), Toronto, Canada, May 2002
8. Curbera, F., Duftler, M., Khalaf, R., Mukhi, N., Nagy, W., Weerawarana, S.: Unraveling the Web Services Web - An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, Vol.6 Issue 2, (2002) 86-93
9. R. Kanawati. Licra: a replicated-data management algorithm for distributed synchronous groupware application. Parallel Computing, Vol. 22, (1997) 1733-1746.
10. Y. Yang, C. Sun, Y. Zhang, and X. Jia: Real-time cooperative editing on the Internet. IEEE Internet Computing, May/June, (2000).

Author Index