

ated to depict how the interface responds to user interaction. Content objects should be identified (even if they have not yet been designed and developed), WebApp functionality should be shown, and navigation links should be indicated.

6. **Refine interface layout and storyboards using input from aesthetic design.** Rough layout and storyboarding is completed by Web engineers, but the aesthetic look and feel for a major commercial site is often developed by artistic, rather than technical, professionals.
7. **Identify user interface objects that are required to implement the interface.** This task may require a search through an existing object library to find those reusable objects (classes) that are appropriate for the WebApp interface. In addition, any custom classes are specified at this time.
8. **Develop a procedural representation of the user's interaction with the interface.** This optional task uses UML sequence diagrams and/or activity diagrams (discussed in Chapter 18) to depict the flow of activities (and decisions) that occur as the user interacts with the WebApp.
9. **Develop a behavioral representation of the interface.** This optional task makes use of UML state diagrams (discussed in Chapter 18) to represent state transitions and the events that cause them. Control mechanisms (i.e., the objects and actions available to the user to alter a WebApp state) are defined.
10. **Describe the interface layout for each state.** Using design information developed in Tasks 2 and 5, associate a specific layout or screen image with each WebApp state described in Task 9.
11. **Refine and review the interface design model.** Review of the interface should focus on usability (Chapter 12).

It is important to note that the final task set chosen by a Web engineering team must be adapted to the special requirements of the application that is to be built.



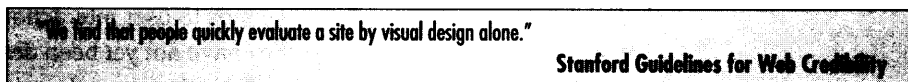
## 19.4 AESTHETIC DESIGN



*Not every Web engineer (or software engineer) has artistic (aesthetic) talent. If you fall into this category, hire an experienced graphic designer for aesthetic design work.*

Aesthetic design, also called *graphic design*, is an artistic endeavor that complements the technical aspects of Web engineering. Without it, a WebApp may be functional, but unappealing. With it, a WebApp draws its users into a world that embraces them on a visceral, as well as an intellectual, level.

But what is aesthetic? There is an old saying, “beauty exists in the eye of the beholder.” This is particularly appropriate when aesthetic design for WebApps is considered. To perform effective aesthetic design, we again return to the user hierarchy developed as part of the analysis model (Chapter 18) and ask, who are the WebApp’s users and what “look” do they desire?



### 19.4.1 Layout Issues

Every Web page has a limited amount of “real estate” that can be used to support non-functional aesthetics, navigation features, information content, and user-directed functionality. The “development” of this real estate is planned during aesthetic design.

Like all aesthetic issues, there are no absolute rules when screen layout is designed. However, a number of general layout guidelines are worth considering:

*Don't be afraid of white space.* It is inadvisable to pack every square inch of a Web page with information. The resulting clutter makes it difficult for the user to identify needed information or features and creates visual chaos that is not pleasing to the eye.

*Emphasize content.* After all, that's the reason the user is there. Nielsen [NIE00] suggests that the typical Web page should be 80 percent content with the remaining real estate dedicated to navigation and other features.

*Organize layout elements from top-left to bottom-right.* The vast majority of users will scan a Web page in much the same way as they scan the page of a book—top-left to bottom-right.<sup>8</sup> If layout elements have specific priorities, high-priority elements should be placed in the upper-left portion of the page real estate.

*Group navigation, content, and function geographically within the page.* Humans look for patterns in virtually all things. If there are no discernable patterns within a Web page, user frustration is likely to increase (due to unnecessary searching for needed information).

*Don't extend your real estate with the scrolling bar.* Although scrolling is often necessary, most studies indicate that users would prefer not to scroll. It is better to reduce page content or to present necessary content on multiple pages.

*Consider resolution and browser window size when designing layout.* Rather than defining fixed sizes within a layout, the design should specify all layout items as a percentage of available space [NIE00].

### 19.4.2 Graphic Design Issues

*Graphic design* considers every aspect of the look and feel of a WebApp. The graphic design process begins with layout (Section 19.4.1) and proceeds into a consideration of global color schemes, typefaces, sizes, and styles, the use of supplementary media (e.g., audio, video, animation), and all other aesthetic elements of an application. The interested reader can obtain design tips and guidelines from many Web sites that

<sup>8</sup> There are exceptions that are cultural and language-based, but this rule does hold for most users.

are dedicated to the subject (e.g., [www.graphic-design.com](http://www.graphic-design.com), [www.grantasticdesigns.com](http://www.grantasticdesigns.com), [www.wpdfd.com](http://www.wpdfd.com)) or from one or more print resources (e.g., [BAG01], [CLO01], or [HEI02]).

## INFO



### Well-Designed Web Sites

Sometimes, the best way to understand good WebApp design is to look at a few examples. In his article, "The Top Twenty Web Design Tips," Marcelle Toor (<http://www.graphic-design.com/Web/feature/tips.html>) suggests the following Web sites as examples of good graphic design:

[www.primo.com](http://www.primo.com)—a design firm headed by Primo Angeli.  
[www.workbook.com](http://www.workbook.com)—this site showcases work by illustrators and designers.

[www.pbs.org/riverofsong](http://www.pbs.org/riverofsong)—a television series for public TV and radio about American music.

[www.RKDINC.com](http://www.RKDINC.com)—a design firm with an on-line portfolio and good design tips.

[www.commarts.com/career/index.html](http://www.commarts.com/career/index.html)—*Communication Arts* magazine, a trade periodical for graphic designers. A good source for other well-designed sites.

[www.btdnyc.com](http://www.btdnyc.com)—a design firm headed by Beth Toudreau.

## 19.5 CONTENT DESIGN

*Content design* focuses on two different design issues, each addressed by individuals with different skill sets. Content design develops a design representation for content objects and represents the mechanisms required to instantiate their relationships to one another. This design activity is conducted by Web engineers.

In addition, content design is concerned with the representation of information within a specific content object—a design activity that is conducted by copywriters, graphic designers, and others who generate the content to be used within a WebApp.

*"Good designers can create normalcy out of chaos; they can clearly communicate ideas through the organizing and manipulating of words and pictures."*

**Jeffery Veen**

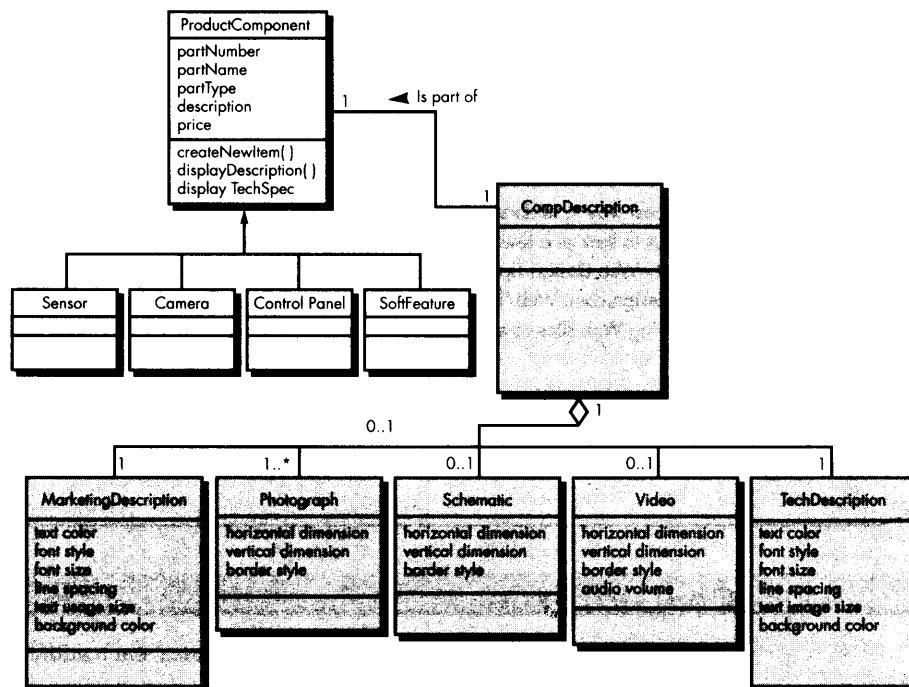
### 19.5.1 Content Objects

The relationship between content objects defined as part of the WebApp analysis model (e.g., Figure 18.3) and design objects representing content is analogous to the relationship between analysis classes and design components described in Chapter 11. In the context of Web engineering, a *content object* is more closely aligned with a data object for conventional software. A content object has attributes that include content specific information (normally defined during WebApp analysis modeling) and implementation specific attributes that are specified as part of design.

As an example, consider the analysis class developed for the *SafeHome* e-commerce system named **ProductComponent** that was developed in Chapter 18 and represented as shown in Figure 19.4. In Chapter 18, we noted an attribute **description**

FIGURE 19.4

Design representation of content objects



that is represented here as a design class named **CompDescription** composed of five content objects: **MarketingDescription**, **Photograph**, **TechDescription**, **Schematic**, and **Video** shown as shaded objects noted in the figure. Information contained within the content object is noted as attributes. For example, **Photograph** (a .jpg image) has the attributes **horizontal dimension**, **vertical dimension**, and **border style**.

UML association and an aggregation<sup>9</sup> may be used to represent relationships between content objects. For example, the UML association shown in Figure 19.4 indicates that one **CompDescription** is used for each instance of the **ProductComponent** class. **CompDescription** is composed of the five content objects shown. However, the multiplicity notation shown indicates that **Schematic** and **Video** are optional (0 occurrences are possible), one **MarketingDescription** and **TechDescription** is required, and one or more instances of **Photograph** is used.

### 19.5.2 Content Design Issues

Once all content objects are modeled, the information that each object is to deliver must be authored and then formatted to best meet the customer's needs. Content authoring is the job of specialists who design the content object by providing an outline of information to be delivered and an indication of the types of generic content

<sup>9</sup> Both of these representations are discussed in Chapter 8.



*Users tend to tolerate vertical scrolling more readily than horizontal scrolling. Avoid wide page formats.*

objects (e.g., descriptive text, graphic images, photographs) that will be used to deliver the information. Aesthetic design (Section 19.4) may also be applied to represent the proper look and feel for the content.

As content objects are designed, they are “chunked” [POW00] to form WebApp pages. The number of content objects incorporated into a single page is a function of user needs, constraints imposed by download speed of the Internet connections, and restrictions imposed by the amount of scrolling that the user will tolerate.

## 19.6 ARCHITECTURE DESIGN

*Architecture design* is tied to the goals established for a WebApp, the content to be presented, the users who will visit, and the navigation philosophy that has been established. The architectural designer must identify content architecture and WebApp architecture. *Content architecture*<sup>10</sup> focuses on the manner in which content objects (or composite objects such as Web pages) are structured for presentation and navigation. *WebApp architecture* addresses the manner in which the application is structured to manage user interaction, handle internal processing tasks, effect navigation, and present content.

**“The architectural structure of a well designed site is not always apparent to the user—nor should it be.”**

**Thomas Powell**

In most cases, architecture design is conducted in parallel with interface, aesthetic, and content design. Because the WebApp architecture may have a strong influence on navigation, the decisions made during this design activity will influence work conducted during navigation design.

### 19.6.1 Content Architecture

The design of *content architecture* focuses on the definition of the overall hypermedia structure of the WebApp. The design can choose from four different content structures [POW00]:

*Linear structures* (Figure 19.5) are encountered when a predictable sequence of interactions (with some variation or diversion) is common. A classic example might be a tutorial presentation in which pages of information along with related graphics, short videos, or audio are presented only after prerequisite information has been presented. The sequence of content presentation is predefined and generally linear. Another example might be a product order entry sequence in which specific information must be specified in a specific order. In such cases, the structures shown in Figure 19.5 are appropriate. As content and processing become more complex, the purely linear flow shown on the left of the figure gives way to

<sup>10</sup> The term *information architecture* is also used to connote structures that lead to better organization, labeling, navigation, and searching of content objects.

FIGURE 19.5

## Linear structures

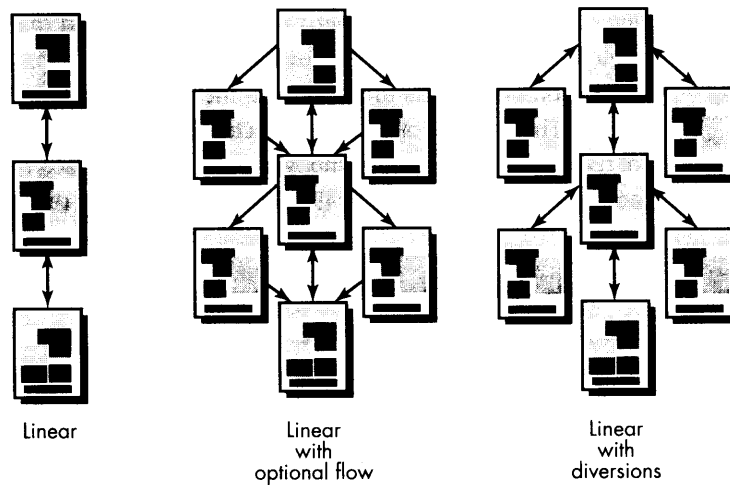
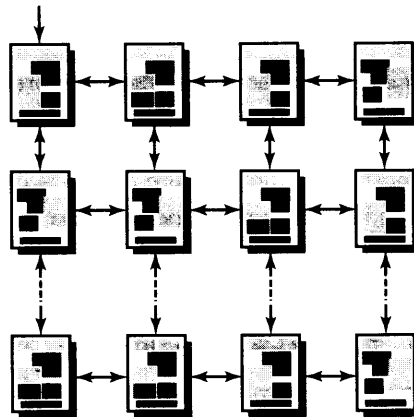


FIGURE 19.6

## Grid structure

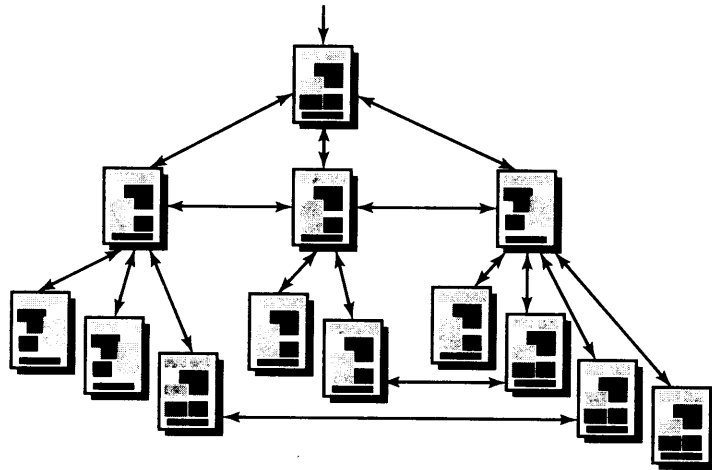


more sophisticated linear structures in which alternative content may be invoked or a diversion to acquire complementary content (structure shown on the right side of Figure 19.5) occurs.

*Grid structures* (Figure 19.6) are an architectural option that can be applied when WebApp content can be organized categorically in two (or more) dimensions. For example, consider a situation in which an e-commerce site sells golf clubs. The horizontal dimension of the grid represents the type of club to be sold (e.g., woods, irons, wedges, putters). The vertical dimension represents the offerings provided by various golf club manufacturers. Hence, a user might navigate the grid horizontally to find the putters column and then vertically to examine the offerings provided by those manufacturers that sell putters. This WebApp architecture is useful only when highly regular content is encountered [POW00].

FIGURE 19.7

Hierarchical structure



*Hierarchical structures* (Figure 19.7) are undoubtedly the most common WebApp architecture. Unlike the partitioned software hierarchies discussed in Chapter 10 that encourage flow of control only along vertical branches of the hierarchy, a WebApp hierarchical structure can be designed in a manner that enables (via hypertext branching) flow of control horizontally, across vertical branches of the structure. Hence, content presented on the far left-hand branch of the hierarchy can have hypertext links that lead to content that exists in the middle or right-hand branch of the structure. It should be noted, however, that although such branching allows rapid navigation across WebApp content, it can lead to confusion on the part of the user.

A *networked* or “*pure web*” structure (Figure 19.8) is similar in many ways to the architecture that evolves for object-oriented systems. Architectural components (in this case Web pages) are designed so that they may pass control (via hypertext links) to virtually every other component in the system. This approach allows considerable navigation flexibility, but at the same time can be confusing to a user.

The architectural structures discussed in the preceding paragraphs can be combined to form *composite structures*. The overall architecture of a WebApp may be hierarchical, but part of the structure may exhibit linear characteristics, while another part of the architecture may be networked. The goal for the architectural designer is to match the WebApp structure to the content to be presented and the processing to be conducted.

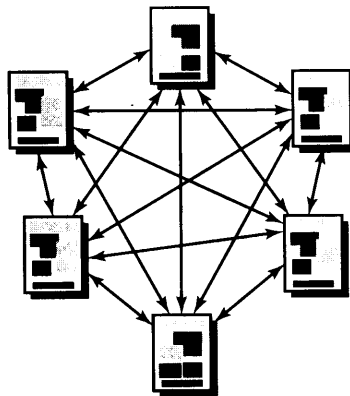
### 19.6.2 WebApp Architecture

*WebApp architecture* describes an infrastructure that enables a Web-based system or application to achieve its business objectives. Jacyntho and his colleagues [JAC02] describe the basic characteristics of this infrastructure in the following manner:

Applications should be built using layers in which different concerns are taken into account; in particular, application data should be separated from the page’s contents

FIGURE 19.8

Network structure



(navigation nodes) and these contents, in turn, should be clearly separated from the interface look-and-feel (pages).

The authors suggest a three-layer design architecture that decouples interface from navigation and from application behavior, and argue that keeping interface, application, and navigation separate simplifies implementation and enhances reuse.

The *Model-View-Controller* (MVC) architecture [KRA88]<sup>11</sup> is one of a number of suggested WebApp infrastructure models that decouples the user interface from the WebApp functionality and informational content. The *model* (sometimes referred to as the “model object”) contains all application specific content and processing logic, including all content objects, access to external data/information sources, and all processing functionality that are application specific. The *view* contains all interface specific functions and enables the presentation of content and processing logic, including all content objects, access to external data/information sources, and all processing functionality required by the end-user. The *controller* manages access to the *model* and the *view* and coordinates the flow of data between them. In a WebApp, “view is updated by the controller with data from the model based on user input” [WMT02]. A schematic representation of the MVC architecture is shown in Figure 19.9.

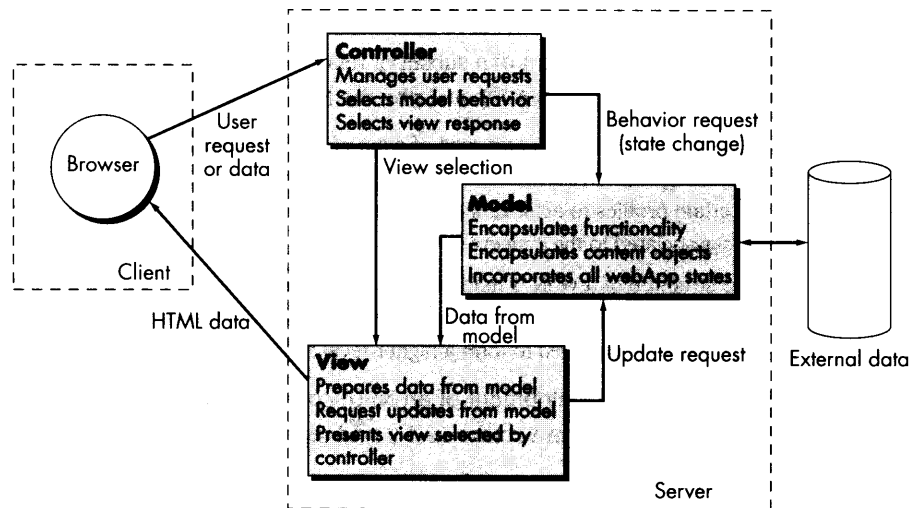
Referring to the figure, user requests or data are handled by the controller. The controller also selects the view object that is applicable based on the user request. Once the type of request is determined, a behavior request is transmitted to the model, which implements the functionality or retrieves the content required to accommodate the request. The model object can access data stored in a corporate database, as part of a local data store or as a collection of independent files. The data developed by the model must be formatted and organized by the appropriate view

### KEY POINT

The MVC architecture decouples the user interface from WebApp functionality and informational content.

<sup>11</sup> It should be noted that MVC is actually an architectural design pattern developed for the Smalltalk environment (see [http://www.cetus-links.org/oo\\_smalltalk.html](http://www.cetus-links.org/oo_smalltalk.html)) and can be used for any interactive application.



**FIGURE 19.9** The MVC architecture (adapted from [JAC02])

object and then transmitted from the application server back to the client-based browser for display on the customer's machine.

In many cases, WebApp architecture is defined within the context of the development environment in which the application is to be implemented (e.g., ASP.net, JWAA, or J2EE). The interested reader should see [FOW03] for further discussion of modern development environments and their role in the design of Web application architectures.

## 19.7 NAVIGATION DESIGN

Once the WebApp architecture has been established and the components (pages, scripts, applets, and other processing functions) of the architecture have been identified, the designer must define navigation pathways that enable users to access WebApp content and functions. To accomplish this, the designer should (1) identify the semantics of navigation for different users of the site, and (2) define the mechanics (syntax) of achieving the navigation.

*"Just wait, Gretel, until the moon rises, and then we shall see the crumbs of bread which I have strewn about, they will show us our way home again."*

from Hansel and Gretel

### 19.7.1 Navigation Semantics

Like many Web engineering activities, navigation design begins with a consideration of the user hierarchy and related use-cases (Chapter 18) developed for each category of user (actor). Each actor may use the WebApp somewhat differently and therefore have different navigation requirements. In addition, the use-cases developed for each actor

will define a set of classes that encompass one or more content objects or WebApp functions. As each user interacts with the WebApp, she encounters a series of *navigation semantic units* (NSUs)—“a set of information and related navigation structures that collaborate in the fulfillment of a subset of related user requirements” [CAC02].

Gnaho and Larcher [GNA99] describe the NSU in the following way:

The structure of a NSU is composed of a set of navigational sub-structures that we call *ways of navigating* (WoN). A WoN represents the best navigation way or path for users with certain profiles to achieve their desired goal or sub-goal. Therefore, the concept of WoN is associated to the concept of User Profile.

The structure of a WoN is made out of a set of relevant *navigational nodes* (NN) connected by *navigational links*, including sometimes other NSUs. That means that NSUs may themselves be aggregated to form a higher-level NSU, or may be nested to any depth.

To illustrate the development of an NSU, consider the use-case, *select SafeHome components*, described in Section 18.1.2 and reproduced here:

**Use-case:** *select SafeHome components*

The WebApp will recommend product components (e.g., control panels, sensors, cameras) and other features (e.g., PC-based functionality implemented in software) for each room and exterior entrance. If I request alternatives, the WebApp will provide them, if they exist. I will be able to get descriptive and pricing information for each product component. The WebApp will create and display a bill-of-materials as I select various components. I'll be able to give the bill-of-materials a name and save it for future reference (see use-case: *save configuration*).

The underlined items in the use-case description represent classes and content objects that will be incorporated into one or more NSUs that will enable a new customer to perform the scenario described in the *select SafeHome components* use-case.

Figure 19.10 depicts a partial semantic analysis of the navigation implied by the *select SafeHome component* use-case. Using the terminology introduced earlier, the figure also represents a way of navigating (WoN) for the SafeHomeAssured.com WebApp. Important problem domain classes are shown along with selected content objects (in this case the package of content objects named **CompDescription**, an attribute of the **ProductComponent** class). These items are navigation nodes. Each of the arrows represents a navigation link<sup>12</sup> and is labeled with the use-initiated action that causes the link to occur.

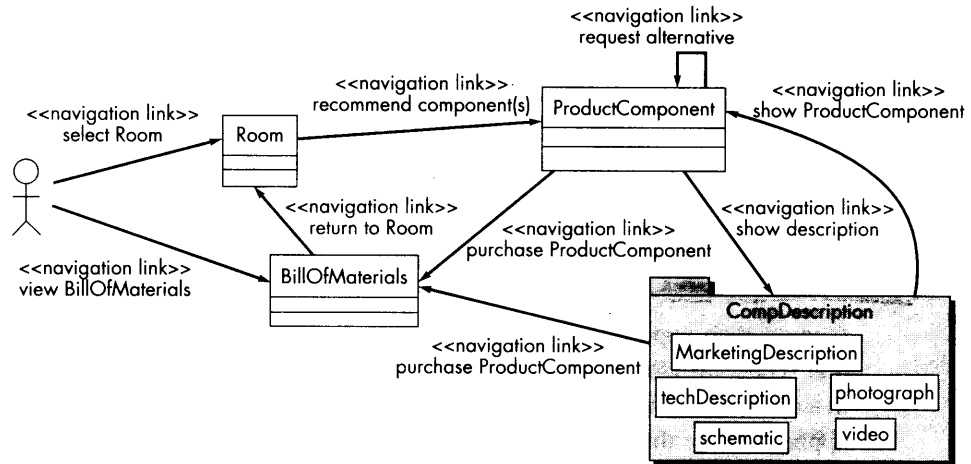
The WebApp designer creates a NSU for each use-case associated with each user role [GNA99]. For example, a **new customer** (Figure 18.1) may have three different use-cases, all resulting in access to different information and WebApp functions. A NSU is created for each goal.

During the initial stages of navigation design, the WebApp content architecture is assessed to determine one or more WoN for each use-case. As noted above, a WoN

## KEY POINT

A NSU describes the navigation requirements for each use-case. In essence, the NSU shows how an actor moves between content objects or WebApp functions.

<sup>12</sup> These are sometimes referred to as *navigation semantic links* (NSL) [CAC02].

**FIGURE 19.10** Creating a NSU

identifies navigation nodes (e.g., content) and the links that enable navigation between them. The WoN are then organized into NSUs.

*"The problem of Web site navigation is conceptual, technical, spatial, philosophical and logistic. Consequently, solutions tend to call for complex improvisational combinations of art, science, and organizational psychology."*

*The Morgan*

### 19.7.2 Navigation Syntax

As design proceeds, the mechanics of navigation are defined. Among many possible options are:



*In most situations, choose either horizontal or vertical navigation mechanisms, but not both.*

- *Individual navigation link*—text-based links, icons, buttons and switches, and graphical metaphors.
- *Horizontal navigation bar*—lists major content or functional categories in a bar containing appropriate links. In general, between four and seven categories are listed.
- *Vertical navigation column*— (1) lists major content or functional categories, or (2) lists virtually all major content objects within the WebApp. If the second option is chosen, such navigation columns can “expand” to present content objects as part of a hierarchy.
- *Tabs*—a metaphor that is nothing more than a variation of the navigation bar or column, representing content or functional categories as tab sheets that are selected when a link is required.
- *Site maps*—provide an all-inclusive table of contents for navigation to all content objects and functionality contained within the WebApp.



*The site map should be accessible from every page. The map itself should be organized so that the structure of WebApp information is readily apparent.*

In addition to choosing the mechanics of navigation, the designer should also establish appropriate navigation conventions and aids. For example, icons and graphical links should look “clickable” by beveling the edges to give the image a three-dimensional look. Audio or visual feedback should be designed to provide the user with an indication that a navigation option has been chosen. For text-based navigation, color should be used to indicate navigation links and to provide an indication of links already traveled. These are but a few of dozens of design conventions that make navigation user-friendly.

---

## 19.8 COMPONENT LEVEL DESIGN

---

Modern Web applications deliver increasingly sophisticated processing functions that (1) perform localized processing to generate content and navigation capability in a dynamic fashion; (2) provide computation or data processing capability that are appropriate for the WebApp’s business domain; (3) provide sophisticated database query and access; (4) establish data interfaces with external corporate systems. To achieve these (and many other) capabilities, the Web engineer must design and construct program components that are identical in form to software components for conventional software.

In Chapter 11, we consider component-level design in some detail. The design methods discussed in Chapter 11 apply to WebApp components with little, if any, modification. The implementation environment, programming languages, and reusable patterns, frameworks, and software may vary somewhat, but the overall design approach remains the same.

---

## 19.9 HYPERMEDIA DESIGN PATTERNS

---

Design patterns that are used in Web engineering encompass two major classes: (1) generic design patterns that are applicable to all types of software (e.g., [BUS96] and [GAM95]) and (2) *hypermedia design patterns* that are specific to WebApps. Generic design patterns have been discussed in Chapter 9. A number of hypermedia patterns catalogs and repositories can be accessed via the Internet.<sup>13</sup>

**“Each pattern is a three-part rule which expresses a relationship between a certain context, a problem, and a solution.”**

**Christopher Alexander**

As we noted earlier in this book, design patterns are a generic approach for solving some small design problem that can be adapted to a much wider variety of specific problems. In the context of Web-based systems German and Cowan [GER00] suggest the following patterns categories:

---

<sup>13</sup> See the sidebar at the end of this section.

**Architectural patterns.** These patterns assist in the design of content and WebApp architecture. Sections 19.6.1 and 19.6.2 present architectural patterns for content and WebApp architecture. In addition, many related architectural patterns are available (e.g., *Java Blueprints* at [java.sun.com/blueprints/](http://java.sun.com/blueprints/)) for Web engineers who must design WebApps in a variety of business domains.

**Component construction patterns.** These patterns recommend methods for combining WebApp components (e.g., content objects, functions) into composite components. When data processing functionality is required within a WebApp, the architectural and component-level design patterns proposed by [BUS96], [GAM95], and others are applicable.

**Navigation patterns.** These patterns assist in the design of NSUs, navigation links, and the overall navigation flow of the WebApp.

**Presentation patterns.** These patterns assist in the presentation of content as it is presented to the user via the interface. Patterns in this category address how to organize user interface control functions for better usability; how to show the relationship between an interface action and the content objects it affects; how to establish effective content hierarchies; and many others.

**Behavior/user interaction patterns.** These patterns assist in the design of user-machine interaction. Patterns in this category address how the interface informs the user of the consequences of a specific action; how a user expands content based on usage context and user desires; how to best describe the destination that is implied by a link; how to inform the user about the status of an on-going interaction and others.

Sources of information on hypermedia design patterns have expanded dramatically in recent years. Interested readers should see [GAR97], [PER99], and [GER00].

## SOFTWARE TOOLS



### Hypermedia Design Patterns Repositories

The IAWiki Web site (<http://iawiki.net/WebsitePatterns>) is a collaborative discussion space for information architects that contains many useful resources. Among them are links to a number of useful hypermedia patterns catalogs and repositories. Hundreds of design patterns are represented:

**Hypermedia Design Patterns Repository**

<http://www.designpattern.lu.unisi.ch/>

**InteractionPatterns by Tom Erickson**

[http://www.pliant.org/personal/Tom\\_Erickson/InteractionPatterns.html](http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html)

**Web Design Patterns by Martijn vanWelie**

<http://www.welie.com/patterns/>

**Improving Web Information Systems with Navigational Patterns**

<http://www8.org/w8-papers/5b-hypertext-media/improving/improving.html>

**An HTML 2.0 Pattern Language**

<http://www.anamorph.com/docs/patterns/default.html>

**Common Ground**

[http://www.mit.edu/~jtiddwell/interaction\\_patterns.html](http://www.mit.edu/~jtiddwell/interaction_patterns.html)

**Patterns for Personal Web Sites**

<http://www.rdrop.com/~half/Creations/Writings/Web.patterns/index.html>

**Indexing Pattern Language**

<http://www.cs.brown.edu/~rms/InformationStructures/Indexing/Overview.html>

## 19.10 OBJECT-ORIENTED HYPERMEDIA DESIGN METHOD (OOHDM)





A number of design methods for Web applications have been proposed over the past decade. To date, no single method has achieved dominance. In this section we present a brief overview of one of the most widely discussed WebApp design methods—OOHDM.<sup>14</sup>

*Object-Oriented Hypermedia Design Method (OOHDM)* was originally proposed by Daniel Schwabe and his colleagues [SCH95, SCH98]. OOHDM is composed of four different design activities: conceptual design, navigational design, abstract interface design, and implementation. A summary of these design activities is shown in Figure 19.11 and discussed briefly in the sections that follow.

### 19.10.1 Conceptual Design for OOHDM

OOHDM *conceptual design* creates a representation of the subsystems, classes, and relationships that define the application domain for the WebApp. UML may be used<sup>15</sup> to create appropriate class diagrams, aggregations and composite class representa-

**FIGURE 19.11** Summary of the OOHDM method (adapted from [SCH95])

|                   |  Conceptual design |  Navigational design |  Abstract interface design |  Implementation |
|-------------------|---|---|---|--|
| Work products     | Classes, sub-systems, relationships, attributes   | Nodes, links, access structures, navigational contexts, navigational transformations                    | Abstract interface objects, responses to external events, transformations                                     | executable WebApp  |
| Design mechanisms | Classification, composition, aggregation, generalization, specialization                              | Mapping between conceptual and navigation objects   | Mapping between navigation and perceptible objects  | Resource provided by target environment  |
| Design concerns   | Modeling semantics of the application domain  | Takes into account user profile and task. Emphasis on cognitive aspects.                                | Modeling perceptible objects, implementing chosen metaphors. Describe interface for navigational objects      | Correctness; Application performance; completeness.  |

<sup>14</sup> A comprehensive comparison of 10 hypermedia design methods has been developed by Koch [KOC99].

<sup>15</sup> OOHDM does not prescribe a specific notation; however, the use of UML is common when this method is applied.