Chapter 12

Database Systems

Multimedia database systems are database systems where, besides text and other discrete data, audio and video information will also be stored, manipulated and retrieved. To provide this functionality, multimedia database systems require a proper storage technology and file system.

Current storage technology allows for the possibility of reading, storing and writing audio and video information in real-time. This can happen either through dedicated external devices, which have long been available, or through system integrated secondary storage. The external devices were developed for studio and electronic entertainment applications; they were *not* developed for storage of discrete media. An example is a video recorder controlled through a digital interface. On the other hand, the integrated secondary storage for audio and video is often based on optical technology (see Chapter 7). The system integration of secondary storage is performed by the operating system through the device drivers. ross@systems.seas.upenn.edu Further, multimedia databases need proper file systems (see Chapter 9) because external devices can be accessed easier through a file directory. The same applies to the integrated secondary storage.

12.1 Multimedia Database Management System

Multimedia applications often address file management interfaces at different levels of abstraction. Consider the following three applications: a *hypertext application* can manipulate nodes and edges; an *audio editor* can read, write and manipulate audio data (sentences); an *audio-video distribution service* can distribute stored video information.

At first, it appears that these three applications do not have much in common, but in all three their functions can uniformly be performed using a *Multimedia DataBase Management System (MDBMS).* The reason is that in general, the main task of a *DataBase Management System (DBMS)* is to abstract from the details of the storage access and its management. As shown in Figure 1.1 (architectural overview of all multimedia system components), MDBMS is embedded in the multimedia system domain, located between the application domain (applications, documents) and the device domain (storage, compression and computer technology). The MDBMS is integrated into the system domain through the operating system and communication components. Therefore, all three applications can be put on the same abstraction level with respect to DBMS. Further, a DBMS provides other properties in addition to storage abstraction:

- *Persistence of Data*

  The data may outlive processing programs, technologies, etc. For example, insurance companies must keep data in databases for several decades. During this time, computer technology advances and with these changes operating systems and other programs advance. This implies that a DBMS should be able to manipulate data even after the changes of the surrounding programs.

- *Consistent View of Data*

  In multi-user systems it is important to provide a consistent view of data during processing database requests at certain points. This property is achieved using *time synchronization protocols.*

- *Security of Data*

Security of data and integrity protection in databases in case of system failure is one of the most important requirements DBMS. This property is provided using the *transaction concept.*

- *Query and Retrieval of Data*

  Different information (entries) is stored in databases, which later can be retrieved through database queries. Database queries are formulated with *query languages* such as SQL. Further, every entry in a database includes its own state information (e.g., entry was modified), which needs to be retrieved correctly to provide correct information about the entry.

The inclusion of multimedia functionalities entails database management of very large amounts of data distributed over different secondary storage media. For example, in the case of a relational database, a vector of attributes representing a video clip with sound can require a storage capacity of several megabytes. Besides the need for a large storage capacity, real-time requirements are also demanded of DBMS processing of continuous data. The design of an MDBMS must comply with these requirements. It is important to point out that if external devices are used, the real-time requirements are outside of the DBMS scope and the connected devices implicitly meet them. In an integrated system, real-time requirements need to be considered inside of the DBMS scope. A more detailed consideration of MDBMSs and multimedia systems is given in [Mey91].

In the following sections, the tasks of a database system with respect to multimedia and an overview of system architecture are presented. Further, a description of multimedia data formats and their related operations follows. Final comments of this chapter concentrate on the expected research results in this area.

12.2 Characteristics of an MDBMS

An MDBMS can be characterized by its objectives when handling multimedia data:

1. *Corresponding Storage Media*

Multimedia data must be stored and managed according to the specific characteristics of the available storage media. Here, the storage media can be both computer integrated components and external devices. Additionally, read-only (such as a CD-ROM), write-once and write-many storage media can be used.

2. *Descriptive Search Methods*

   During a search in a database, an entry, given in the form of text or a graphical image, is found using different queries and the corresponding search methods. A query of multimedia data should be based on a *descriptive, content-oriented search* in the form, for example, of "The picture of the woman with a red scarf". This kind of search relates to all media, including video and audio.

3. *Device-independent Interface*

   The interface to a database application should be device-independent. For example, a parameter could specify that the following audio and video data will not change in the future. Hence, the MDBMS will know that the storage media is a *CD-WO*.

4. *Format-independent Interface*

   Database queries should be independent from the underlying media format, meaning that the interfaces should be format-independent. The programming itself should also be format-independent, although in some cases, it should be possible to access details of the concrete formats. Hence, the application programmer determines the level of format abstraction with its advantages and disadvantages. This leads to the development of applications which may also make use of future formats. MDBMS device-independence and format-independence allow for new storage technologies to be used without changing the current multimedia database applications.

5. *View-specific and Simultaneous Data Access*

   The same multimedia data can be accessed (even simultaneously) through different queries by several applications. Hence, consistent access to shared data (e.g., shared editing of a multimedia document among several users) can be implemented.

6. *Management of Large Amounts of Data*

   The DBMS must be capable of handling and managing large amounts of data and satisfying queries for individual relations among data or attributes of relations [Fel90].

7. *Relational Consistency of Data Management*

   Relations among data of one or different media must stay consistent corresponding to their specification. The MDBMS manages these relations and can use them for queries and data output. Therefore, for example, *navigation* through a document is supported by managing relations among individual parts of a document. There are several different relations [Mey91]:

   - The *attribute relation* provides different descriptions or presentations of the same object. For example, consider a "bird" entry in an ornithological lexicon. Here, the voice for each bird is recorded as an audio signal, the flight of each bird is presented as a motion video and additional images and text accompany the bird description. From the viewpoint of a relational database, an attribute vector is assigned to each bird, including the different representations as attributes.
   - The *component relation* includes all the parts belonging to a data object. A catalogue of the components of a car and a book, consisting of chapter, section and subsection components, are examples of this relation.
   - The *substitution relation* defines different kinds of presentation of the same information. For example, the results of an equation can be presented as tables, graphs or animation.
   - A *synchronization relation* describes the temporal relations between data units. Lip synchronization between audio and video is one example of this relation.

8. *Real-time Data Transfer*

   The read and write operations of continuous data must be done in real-time. The data transfer of continuous data has a higher priority than other database management actions. Hence, the primitives of a multimedia operating system should be used to support the real-time transfer of continuous data.

9. *Long Transactions*

   The performance of a transaction in a MDBMS means that transfer of a large amount of data will take a long time and must be done in a reliable fashion. An example of a long transaction is the retrieval of a movie.

Most of the current prototypes of an MDBMS include the management of storage, local multimedia devices and adapters. This pragmatic approach has the advantage that a multimedia system is fully managed in agreement with the requirements of the considered DBMS. This approach is also often chosen for document and hypertext processing systems. Note that, for example, the results from the network communication (e.g., management of remote devices) are not often taken into account in an MDBMS. However, the remote multimedia devices in MDBMS must be managed too. The current solution is that the device management, as well as the integration of local and remote devices, is often developed in conjunction with the communication system.

In both research areas (MDBMS and communication systems), the development of multimedia primitives as part of the operating system is not sufficiently considered, although the management of multimedia devices should actually be the task of the operating system. In our architecture model, shown in Figure 1.1, the system components around MDBMS and MDBMS itself have the following functions:

- The operating system provides the management interface for MDBMS to all local devices.
- The MDBMS provides an abstraction of the stored data and their equivalent devices, as is the case in DBMS without multimedia.
- The communication system provides for MDBMS abstractions for communication with entities at remote computers. These communication abstractions are specified through interfaces according to, for example, the Open System Interconnection (OSI) architecture.
- A layer above the DBMS, operating system and communication system can unify all these different abstractions and offer them, for example, in an object-

oriented environment such as a *toolkit*. Thus, an application should have access to each abstraction at different levels.

This model corresponds, more or less, to the proposed MDBMS architecture presented in [Mey91]. To refine this architecture, for each medium, an individual *management unit* could be implemented, which would be used by the medium-specific server, and the *services* of the management unit could be offered to a common query processor [Loc90]. This query processor is managed by a *database application interface manager*. The application interface manager provides an application interface to the user with a specification of different *views* to a database entry (e.g., entry "bird" can be specified by sound, motion video and/or text views). Another management unit processes the relations between data of different media.

This MDBMS architecture does not imply any implementation with the same partition of the components.

12.3 Data Analysis

Meyer-Wegener performed a detailed analysis of multimedia data storage and manipulation in a multimedia database system [Mey91]. This section describes the analysis of compressed data and other media. In the analysis, two questions are addressed:

1. **How are these data structured?**

   It is important to specify what kind of information is needed in the entry structure to process the multimedia entry in a MDBMS.

2. **How can these data be accessed?**

   That is to say, how are the proper operations defined to access multimedia entries. One can define media-dependent, as well as media-independent operations. In a next step, a class hierarchy with respect to object-oriented programming may be implemented.

In the following section we will consider the first question. The second question is analyzed in detail in Section 12.5.

12.4 Data Structure

In general, data can be stored in databases either in unformatted (unstructured) form or in formatted (structured) form. *Unformatted* or *unstructured* data are presented in a unit where the content cannot be retrieved by accessing any structural detail. For example, a data description such as "Mr. Clemens Engler is a student in the eighth term" cannot be accessed by any structural detail.

Formatted or *structured* data are stored in variables, fields or attributes with corresponding values. Here, the particular data parts are characterized according to their properties. For example, a data description such as:

```
A.Student.Surname = Engler

A.Student.GivenName = Clemens

A.Term = 8
```

can be accessed by structural details (student's given name, surname or term).

Additionally, multimedia data can be stored in databases as *raw*, *registering* and *descriptive data types.*

12.4.1 Raw Data

An uncompressed image consists of a set of individual pixels. The pixels represent *raw data* in the form of bytes and bits. They create the unformatted information units, which represent a long sequence or set of symbols, pixels, sample values, etc.

12.4.2 Registering Data

To retrieve and correctly interpret such an image, the details of the coding and the size of the image must be known. Let us assume that each pixel in an image is encoded with eight bits for the luminance and both chrominance difference signals. The resolution will be $1,024 \times 1,024$ pixels. These registering data are necessary to provide a correct interpretation of the raw data. Traditional DBMSs usually know only numbers and characters, which have fixed semantics; therefore, no additional description is required. Image, audio and video data allow for a number of attributes during coding and structuring. Without this additional description, the multimedia data could only be interpreted with difficulty, or not at all. Many components and applications of a multimedia system assume an implicit knowledge which in individual cases may also be sufficient, but MDBMS should be provided as a generally accessible service for all applications and components. Therefore, it is necessary to also handle all different formats. Moreover the semantics of the registering data can be extended to define existing relations between data objects of one or several media.

12.4.3 Descriptive Data

Today, the search for textual and numerical content is very effective. However, the search for image, audio or video information is much more difficult. Therefore, optional description (descriptive data) should be assigned to each multimedia unit. In the case of an image, the particular scene could be described in the form of text. These descriptive data provide additional redundant information and ease data retrieval during later searches. Descriptive data could be presented in unstructured or structured form.

12.4.4 Examples of Multimedia Structures

We present examples of raw, registered, and descriptive data for different media such as text, image, video and audio.

In the case of *text*, the individual forms are:

1. Characters represent raw data.
2. The registering data describe the coding (e.g., ASCII). Additionally, a length entry must follow or an end symbol must be defined.
3. The descriptive data may include information for layout and logical structuring of the text or keywords.

Images can be stored in databases using the following forms:

1. Pixels (pixel matrix) represent raw data. A compressed image may also consist of a *transformed pixel* set. For example, the coefficients of the discrete cosine transformation in two-dimensional frequency presentation represent a *transformed pixel* set. A further compression of the raw data can include a set of entropy-encoded data.
2. The registering data include the height and width of the picture. Additionally, the details of coding are stored here. For example, in the case of a JPEG compressed image, the mode is entered first. This may be a specific JPEG mode based on a discrete cosine transformation. Additionally, for example, the eight bits per sample value for image processing, the sequential image structure and the entropy encoding scheme are defined. The tables for quantizing process and entropy encoding must also be specified.
3. Examples of descriptive data are individual lines, surfaces and subjects, or situations as a whole scene (e.g., "Birthday and 1995 New Year's Eve celebration at Lisa's favorite restaurant", or

```
B.Reason = New Year's Eve / Birthday
B.Date = 12/31/95
B.Place = Favorite Restaurant
B.Name = Lisa
B.Keyword1 = New Year's Eve Celebration
```

A motion video sequence can have a very different set of characteristics. It consists of the following information:

1. Raw data are defined in the simplest case through a sequence of pixel matrices. Mostly through motion video coding, the redundancies over several images are used for data reduction (intra-frame coding), so that each image does not carry all the necessary data for decompression. Also, a variable-rate data stream can be created.

2. The registering data provide, in addition to other information, the number of images per second (see Chapter 6). A data stream, coded according to the CCITT H.261 standard, is described as being QCIF (Quarter Common Intermediate Format) with a resolution of 177 x 144 pixels for the luminance component and 88 x 72 pixels for the color difference components. The motion video, coded according to MPEG-2, is described by the relation between consecutive images; types are coded (1 I-frame, 2 B-frame, 1 P-frame, 2 P-frame, 1 I-frame, etc.). Random access to each individual image of the motion video must follow.

3. The descriptive data provide a scene description (e.g., "Jan's birthday party with his friends from kindergarten").

Individual *audio* sequences can be classified according to the following scheme:

1. Raw data may be the digital sample values created by a simple PCM coding. The compressed values may also be considered as raw data.

2. The registering data represent the properties of the audio coding. Using a PCM coding, the sample rate, the quantization line and the resolution of the individual samples are the registering data. Compressed audio data can also carry additional information used by a parameterized decoder. Often, the coding information is already included in the raw data (e.g., ADPCM coding).

3. The descriptive data represent the content of the audio passages in a short form. In the case of a music composition, the name of the composition, the composer name and the name of the player can be entered. In the case of speech, a short content description, or the whole text can be written down.

12.4.5 Comments on Data Analysis

Audio and video data are often stored in a composed, integrated form (e.g., by using MPEG). This guarantees a continuous data stream during the output process. The recording can also be a combination of individual audio and video data. A hierarchy in the media can be inserted. Hence, DBMS can address combined media in the form of raw, registering and descriptive data.

For the application, a format-independent access is important and should be supported by a DBMS. Therefore, it makes sense to define access to uncompressed data at the application interface, although the data are actually processed (e.g., stored, transmitted) in a compressed form.

The division of data into raw, registering and descriptive data types requires strict management during database manipulation. For this management, system support and generation of context-dependent frames for text description should be implemented. If, for example, a motion video of landscapes is used for a travel catalogue, and some data already exist, the structure of the new descriptive data should be derived from the existing descriptive data of other entries.

12.5 Operations on Data

An MDBMS must offer, for all data types presented in Section 12.4, corresponding *operations* for archival and retrieval. The media-related operations will be handled as part of or an extension of query languages (e.g., SQL). In databases, following different classes of operations for each medium are needed: input, output, modification, deletion, comparison and evaluation.

The *input (insert/record)* operation means that data will be written to the database. In this case, the raw and registering data are always needed. The descriptive data can be attached later. If during the input operation of motion video and audio the length of the data is not known a priori, the MDBMS may have problems choosing the proper server or disk.

The *output (play)* operation reads the raw data from the database according to the

registering data. Hence, for decoding a JPEG coded image, the Huffman table can be transmitted to the decoder in advance. The transmission of the raw data follows.

Modification usually considers the raw data. The modification of image data should be done by an editor. For motion video, cutting with *in/out fading* is usually needed. For audio data, in addition to in/out-fading, the volume, bass, treble and eventually balance can also be modified. The modification attributes are stored in registering data. Here, the attributes are defined as time-dependent functions performed during play of data. Modification can also be understood as a data conversion from one format to another. In this case, the registering data must be modified together with raw data. Another variant of modification is transformation from one medium to another, such as text-to-speech transformation. The conversion function, analogous to an editor, should be implemented outside the MDBMS. Such a transformation is implemented through reading of the data, externally converting it to another medium and recording the transformed data in the database.

During the *delete* operation, the consistency of the data must be preserved, i.e., if raw data of an entry are deleted, all other data types (registering and descriptive data) related to raw data are deleted.

Many queries to the MDBMS consist of a search and retrieval of the stored data. These queries are based on *comparison* information. Here, individual patterns in the particular medium are compared with the stored raw data. This kind of search is not very successful. Another approach uses pattern recognition where a pattern from raw data may be stored as registering data and a comparison is based on this pattern. The current efficiency of this approach is low for MDBMSs and is only used for certain applications. A comparison can also be based on the corresponding format of the descriptive data. Here, each audio sequence can be identified according to its unambiguous name (a maximum of 16 characters) and the creation time.

Other comparisons are based on content-oriented descriptive data. For example, the user enters the nominal phrase with a limited set of words. The MDBMS converts this input into predicates. In this case, synonyms can be used and are managed by the system. This concept allows for a content-related search, which is used for images and can be ported without any difficulties to all types of media [LM89, LM90, Mey91].

The goal of *evaluating* the raw and registering data is to generate the corresponding descriptive data. For example, during the storage of facsimile documents, Optical Character Recognition (OCR) can be used. Otherwise, in most cases, an explicit user input is required. The results in [LM89], [LM90], [Mey91] can be used.

12.6 Integration in a Database Model

A main issue for the implementation and usage of an MDBMS is the *database model*. Primarily, the data types (Section 12.4) and operations (Section 12.5) used are important. These data types and operations can be integrated in both a *relational model* and an *object-oriented model* [SZ87].

Abstract data types of all media can be defined with descriptive attributes according to their formats. For example, consider the attributes of *uncompressed video* in Table 12.1. Attributes are stored in a multimedia database and operations can retrieve and modify them.

Attribute Name	Attribute Type	Attribute Value
height:	integer;	480
width:	integer;	640
encoding:	uncompressed;	YUV
stream_encoding:	s_mode;	PAL
pixel_depth:	integer;	(8,8,8)
rate:	signed integer;	25
colormap_size:	integer;	
colormap:	array;	(...)
pixel:	structure(Y,U,V) of bits	
image:	array (...) of pixels	
video:	timed_list of images	

Table 12.1: *Attributes of uncompressed video.*

12.6.1 Relational Database Model

The simplest possibility to implement a multimedia database is to use the relational database model because the attributes of different media in relational databases are defined in advance. Hence, the attributes can specify not only text (as is done in current database systems), but also, for example, audio or video data types. The main advantage of this approach is its compatibility with current database applications.

In the following paragraphs, we will analyze different types of the relational model using an example. In this example, a relation "student" is given for the admission of a sport institute's students into the obligatory athletics course. A relation's attributes (e.g., picture, exercise devices) can be specified through different media types (e.g., image, motion video). Further, our example database includes other entries such as: "athletics", "swimming" and "analysis".

Student	(Admission_Number	Integer,
	Name	String,
	Picture	Image,
	Exercise_Device_1	Video
	Exercise_Device_2	Video)

Athletics	(Admission_Number	Integer,
	Qualification	Integer,
	The_High_Jump	Video,
	The_Mile_Run	Video)

Swimming	(Admission_Number	Integer,
	Crawl	Video)

Analysis	(Qualification	Integer,
	Error_Pattern	String,
	Comment	Audio)

- *Type 1 Relational Model*

  In the *type 1 relational model*, the value of a certain attribute can be fixed over the particular set of the corresponding attribute types, e.g., the frame rate of motion video can be fixed.

  For our example it means that when attributes such as exercise devices 1 and 2 from entry "student" are retrieved, the video will play at the fixed rate defined by the type 1 specification.

- *Type 2 Relational Model*

  A variable number of entries can be defined through the type 2 relational model.

  In our example, the individual disciplines (such as athletics and swimming) of each admitted student are identified through their admission numbers.

- *Type 3 Relational Model*

  In addition to the fixed values of attributes per relation and the variable number of entries, an entry can simultaneously belong to several relations. This property is called the type 3 relational model.

  In our example, a video entry of a student performing a high jump can be assigned to the relation "athletics" for qualification purposes as well as to the relation "analysis" for an educational application with an analysis of typical errors in the individual sport disciplines.

12.6.2 Object-oriented Database Model

In object-oriented databases, instead of defining relations, as is the case in relational databases, classes with objects are defined. These objects can be put in relations via a class hierarchy. Therefore, a semantic specialization of classes and objects can follow. For example, if we consider the above example of students from the sport institute, the sport institute is the main class. Different kinds of sport departments (e.g., athletics, swimming) build subclasses of the sport institute class. Students are the objects of each subclass.

Some MDBMSs use this approach with many different kinds of class hierarchies. As a consequence of these many hierarchies, no generally applicable class hierarchy can be recognized today. These systems offer good information navigation and flexible presentation possibilities. On the other hand, in comparison to the relational model, the important set operations for queries (e.g., get an element from a set, include an element into a set) are incompletely supported.

12.7 Comments

MDBMS are still under development. Most MDBS systems are bound very closely to a multimedia application or application area. An example is the current development of MDBMS for multimedia document processing (hypertext, hypermedia). Other application areas, such as conferencing systems, leave any consideration of MDBMS out. Therefore, there are only few application experiences with MDBMS.

If such experiences existed, a *minimal MDBMS* could be defined. According to [Loc90], such a minimal MDBMS is defined as a system which satisfies all requirements put on a MDBMS using the minimal knowledge about the application. This minimal MDBMS has been, until now, unknown.

The database models presented are strongly influenced by the representation of uncompressed images and content-oriented search. Development in data compression techniques and the requirements for higher media quality will bring more frequent use of compressed data.

Attribute types, such as image, video and audio, must be further modified and refined. In this context, the close relation between the data units of one or different media should be integrated.

An implementation of consistency by a transaction [Gra81] and an implementation of composed transactions [Mos82] need to be adapted to the large amounts of data and real-time conditions. These properties would help ensure that a transaction does not have to start again and again. In this case, a predicate with respect to time conditions, as an additional attribute, could be integrated into a transaction.

Research needs to be done in the area of a content-oriented search with respect to continuous media. Queries, such as "Search for the speech probe with the children song about ten small Indians", or "Search for the video clip with Prince Andrew in the Andalusian costume", are difficult to perform. It is still unclear if a solution is actually possible.

With respect to distributed multimedia databases and network access, integration of multimedia communication and database technology is necessary [BCG+90, GCB88, KG89]. An extension of the Remote Database Access (RDA) for continuous data, a distributed MDBMS and a multimedia client-server system need to be implemented. In a client server environment, the concepts of several clients per transaction and/or several servers per transaction should be considered for support of cooperative multimedia applications.

Today, the integration of MDBMS with an operating system is not done in the best way. Most often, time critical data are either processed through separate external devices, or processed with a lower quality through DBMSs. For an integrated MDBMS, harmony with the new operating system extensions must follow.

Chapter 13

Documents, Hypertext and MHEG

A *document* consists of a set of structural information that can be in different forms of media, and during presentation can be generated or recorded [App90]. A document is aimed at the perception of a human, and is accessible for computer processing.

13.1 Documents

A *multimedia document* is a document which is comprised of information coded in at least one continuous (time-dependent) medium and in one discrete (time-independent) medium. Integration of the different media is given through a close relation between information units. This is also called *synchronization*. A multimedia document is closely related to its environment of tools, data abstractions, basic concepts and document architecture.

Currently, continuous and discrete data are processed differently: text is processed within an editor program as a type of a programming language (namely the Type Character); a motion picture can be manipulated with the same editor program only through library calls. The goal of abstracting multimedia data is to achieve

integrated, i.e., uniform, description and processing of all media. This reduces the complexity of program generation and maintenance that process multimedia data. Since Chapter 16 discusses in detail different approaches to programming abstractions, we will not concentrate here on this issue. Abstractions of multimedia data serve as the fundamental building block for programming different multimedia applications, especially editors and other document processing tools.

Basic *system concepts* for document processing use multimedia abstractions and also serve as concepts for the information architecture in a document. Thus, we use the terms *document architecture* and *information architecture* interchangeably.

13.1.1 Document Architecture

Exchanging documents entails exchanging the document content as well as the document structure. This requires that both documents have the same *document architecture*. The current standardized, respectively in the progress of standardization, architectures are the *Standard Generalized Markup Language* (SGML) and the *Open Document Architecture* (ODA). There are also proprietary document architectures, such as DEC's *Document Content Architecture* (DCA) and IBM's *Mixed Object Document Content Architecture* (MO:DCA).

Information architectures use their data abstractions and concepts. A document architecture describes the connections among the individual elements represented as *models* (e.g., presentation model, manipulation model). The elements in the document architecture and their relations are shown in Figure 13.1. Figure 13.2 shows a multimedia document architecture including relations between individual discrete media units and continuous media units.

The manipulation model describes all the operations allowed for creation, change and deletion of multimedia information. The representation model defines: (1) the *protocols* for exchanging this information among different computers; and, (2) the *formats* for storing the data. It includes the relations between the individual information elements which need to be considered during presentation. It is important to mention that an architecture may not include all described properties, respectively models.

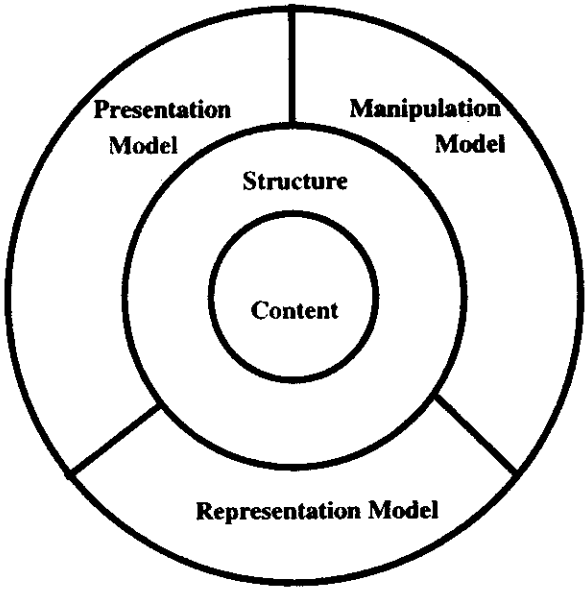

Figure 13.1: *Document architecture and its elements.*

13.1.2 Manipulation of Multimedia Data

The user becomes most aware of multimedia documents through tools for *manipulation of multimedia data*, such as *editors, desktop publishing programs* and other *text processing programs.*

A document undergoes the process shown in Figure 13.3. The information included in a document belongs to a certain document type, e.g., a *business letter* or an *internal memorandum.* The same document can belong to other types which mainly influence the final representation. The transformation from the actual information to its final representation behaves according to rules specific to the document architecture.

The processing cycles (Figure 13.3) of a traditional document and an interactive multimedia presentation are analogous, as shown in Figure 13.4. Currently, an author edits a document with a text editor. Thus, he or she uses the system's character set (e.g., ASCII) as the actual content of a document, as well as a hidden language available in most interactive editors for structural description (e.g.,

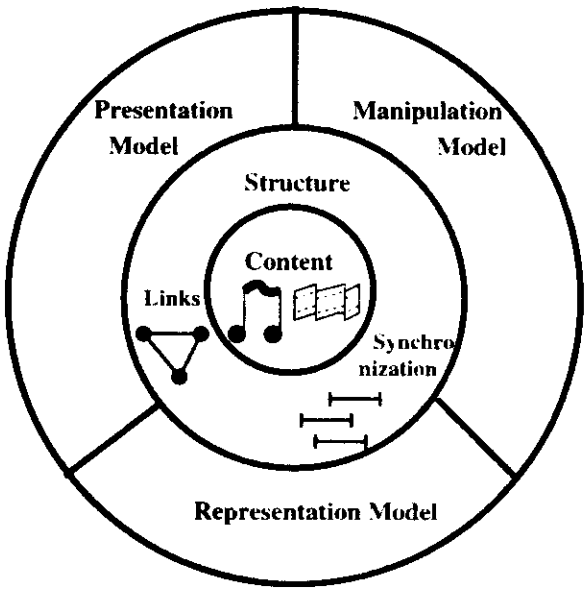

Figure 13.2: *A multimedia document architecture and its constituent elements.*

SGML). At this point, the document exists in a processable representation. The subsequent formatting process determines the layout of the document. The result is a final representation of the document. A typical example of this representation is the typesetting language *PostScript*™. The availability of hypertext and multimedia technology have changed the representation of documents. Although the processing cycle of document generation will remain the same, it is apparent that there will be major changes in how documents are displayed. The output of interactive hypermedia documents will be mostly computer-supported. Therefore, the presentation of a document will have to be not only *final*, but also *executable*. While there are a broad range of processable formats, there are too few final representation formats. It has been internationally recognized that such a final representation (exchange format) is very important, especially in a distributed, heterogeneous system environment. This exchange format for interactive multimedia presentation is called *MHEG (Multimedia and Hypermedia Information Coding Expert Group)*.

Using the main concepts of *hypermedia and hypertext* for multimedia documents, the following sections of this chapter present the document architectures SGML

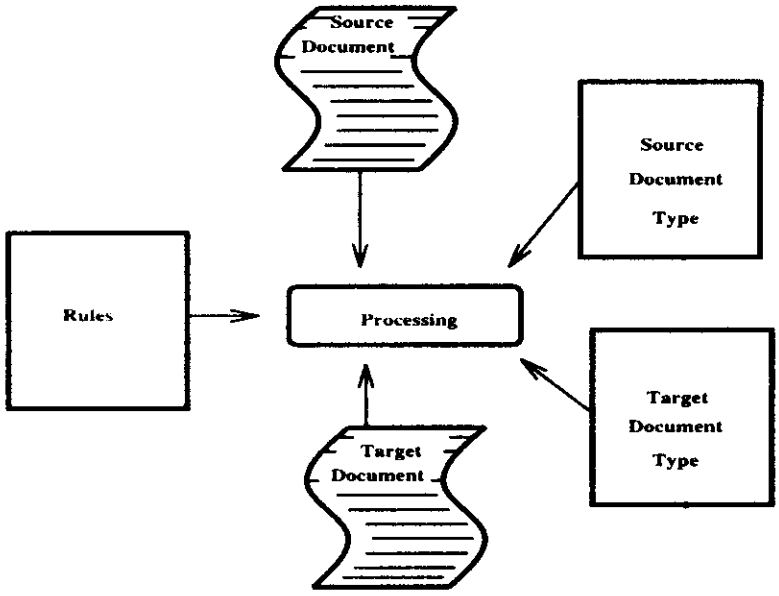

Figure 13.3: *Processing of a document: from the information to the presentation*

and ODA. Finally, MHEG is briefly described.

13.2 Hypertext and Hypermedia

Communication reproduces knowledge stored in the human brain via several media. Documents are one method of transmitting information. Reading a document is an act of reconstructing knowledge. In an ideal case, knowledge transmission starts with an author and ends with a reconstruction of the same ideas by a reader. Information loss is minimal. Figure 13.5 shows this communication process between an author and a reader.

Today's ordinary documents (excluding hypermedia), with their linear form, support neither the reconstruction of knowledge, nor simplify its reproduction. Knowledge must be artificially serialized before the actual exchange. Hence, it is transformed into a *linear* document and the structural information is integrated into the actual content. In the case of hypertext and hypermedia, a graphical structure is possible

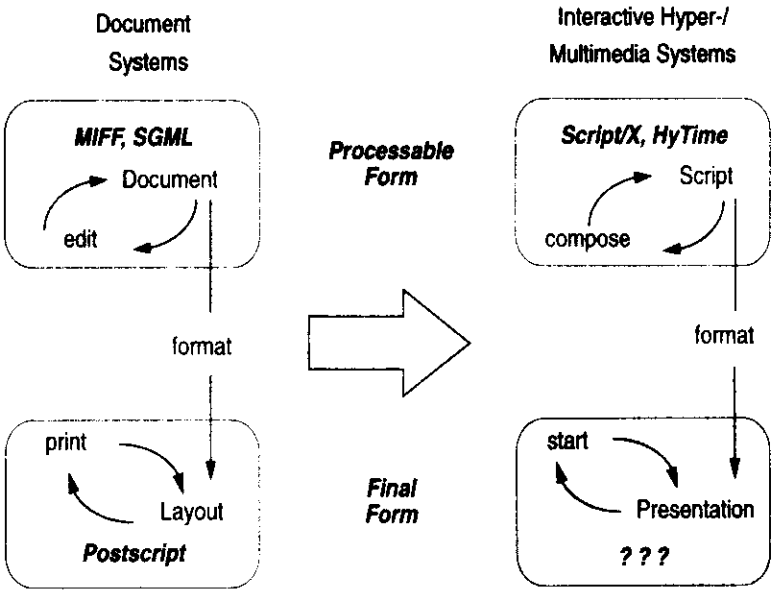

Figure 13.4: *Problem description.*

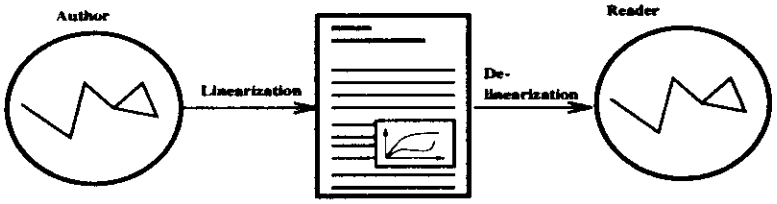

Figure 13.5: *Information transmission [GS90].*

in a document which may simplify the writing and reading processes.

13.2.1 Hypertext, Hypermedia and Multimedia

A book or an article on a paper has a given structure and is represented in a sequential form. Although it is possible to read individual paragraphs without reading previous paragraphs, authors mostly assume a sequential reading. Therefore many paragraphs refer to previous learning in the document. Novels, as well as movies, for example, always assume a pure sequential reception. Scientific literature can

consist of independent chapters, although mostly a sequential reading is assumed. Technical documentation (e.g., manuals) consists often of a collection of relatively independent information units. A lexicon or reference book about the Airbus, for example, is generated by several authors and always only parts are read sequentially. There also exist many cross references in such documentations which lead to multiple searches at different places for the reader. Here, an electronic help facility, consisting of information links, can be very significant.

Figure 13.6 shows an example of such a link. The arrows point to such a relation

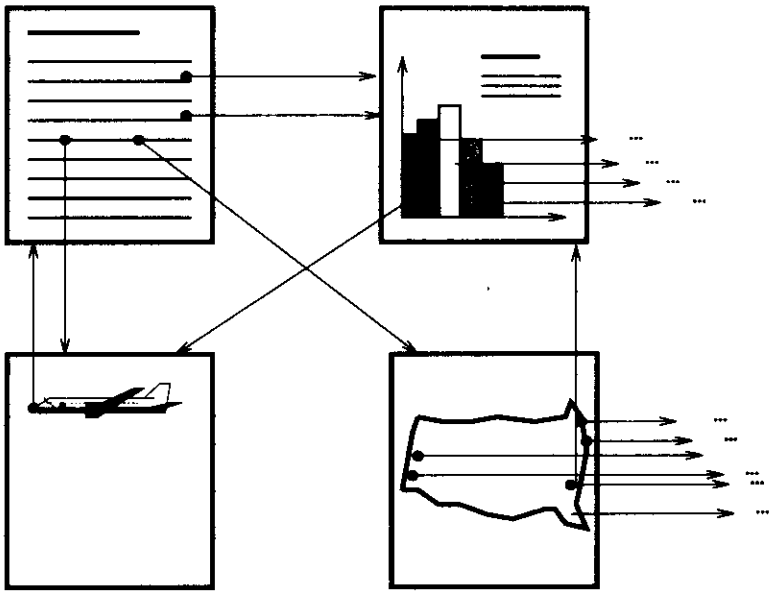

Figure 13.6: *Hypertext data. An example of linking information of different media.*

between the information units (LDUs). In a text (top left in the figure), a reference to the landing properties of aircrafts is given. These properties are demonstrated through a video sequence (bottom left in the figure). At another place in the text, sales of landing rights for the whole USA are shown (this is visualized in the form of a map, using graphics – bottom right in the figure). Further information about the airlines with their landing rights can be made visible graphically through a selection of a particular city. A special information about the number of the different airplanes sold with landing rights in Washington is shown at the top right in the figure with

a bar diagram. Internally, the diagram information is presented in table form. The left bar points to the plane, which can be demonstrated with a video clip.

Non-linear Information Chain

Hypertext and hypermedia have as a major property a *non-linear information link.* There exists not only a reading sequence, but also the reader decides on his/her reading path. The reader can start in a lexicon with a notion *hypertext*, then go through a cross reference to *systems* and finish with a description of *AppleTalk.* By this association, through reference links, the author of the information determines the actual links.

As another example, let us consider this document (our multimedia book) about multimedia systems. The structure of this work consists, besides the introductory and closing chapters (first impression) of a set of *equivalent* chapters. Actually, Chapter 2 has a more important position: it should be read before any of the remaining chapters. The reason is that it includes some fundamentals to explain common notions used in the other chapters. All other chapters are relatively independent and the reader can determine his/her own path. The structure is a tree where the reading path in this linear document is explained verbally and not through the structure. A hypertext structure is a *graph*, consisting of nodes and edges. The references to other chapters and literature citations are, for example, such pointers which build a tree-similar document to a graph.

- The *nodes* are the actual *information units.* They are, for example, the text elements, individual graphics, audio or video LDUs. The information units are shown at the user interface mostly in their own windows.

- The *edges* provide links to other information units. They are usually called *pointers* or *links.* A pointer is mostly a directed edge and includes its own information too.

Anchor

The forward movement in linear sorted documents is called a *navigation* through the graph. At the user interface, the origin of pointers must be marked, so that the user can move to a further information unit. This origin of a pointer is called an *anchor*. A main factor of the user interface is the concept of the anchor: how can the anchor be represented properly?

- A *media-independent representation* can happen through the selection of general graphical elements, such as *buttons*. In such an element, information about the destination node should be included. If the destination node is a text, a short, descriptive text of the content can be represented. In the case of an image, the image content can appear in minimized form on the screen. A visual representation of the video content can follow in form of a *moving icon* (Micon). This is a minimized motion picture which represents a characteristical portion of the video sequence of the destination node (MIT Project Elastic Charles [Bra87]). If the content of the destination node consists of audio information, a visual representation of the audio content must follow. For example, in the case of a music passage, a picture of the composer could be displayed.

- In a *text*, individual words, paragraphs or text sections of different length can be used for representation. The positioning of the pointer to the marked area and double clicking in this area leads to a display of the destination node, connected with the clicked information (e.g., see Figure 13.11).

- In *images*, specific graphical objects or simply areas are defined as selection objects. A specific marking can occur through a color or stripe.

- In a *motion video*, media-independent representations of the anchor are preferred. There can also be time-changing areas used. Mostly, no spatial selection occurs and the particular shown image is conclusive. A timely selection is supported.

- With respect to *audio*, a media-independent solution is used. In this case, a short, descriptive text or an image of the size of an icon is preferably shown.

Figure 13.7 emphasizes the relation among multimedia, hypertext and hypermedia.

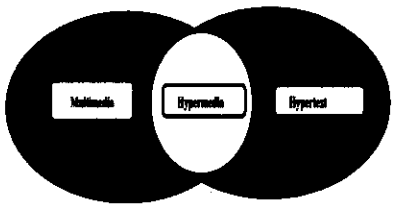

Figure 13.7: *The hypertext, hypermedia and multimedia relationship.*

Hypertext System

A *hypertext system* is mainly determined through non-linear links of information. Pointers connect the nodes. The data of different nodes can be represented with one or several media types. In a pure text system, only text parts are connected. We understand *hypertext* as an information object which includes links to several media.

Multimedia System

A *multimedia system* contains, according to Section 2.3, information which is coded at least in a continuous and discrete medium.

For example, if only links to text data are present, then this is not a multimedia system. It is a *hypertext.* A video conference, with simultaneous transmission of text and graphics, generated by a document processing program, is a multimedia application. Although it does not have any relation to hypertext and hypermedia.

Hypermedia System

As Figure 13.7 shows, a *hypermedia system* includes the non-linear information links of hypertext systems and the continuous and discrete media of multimedia systems.

For example, if a non-linear link consists of text and video data, then this is a hypermedia, multimedia and hypertext system.

As is often the case, we will not use the notion hypermedia in its strongest sense. If not explicitly specified, *hypertext* and *hypermedia* are used interchangeably.

There have been many international conferences covering this area since the late 1980s: Hypertext'87, Hypertext'89, etc. A good overview of articles chosen from the first Hypertext'87 conference is in [ACM88]. ECHT'90 was the first *European Conference on Hypertext* held in Paris. There exists a large number of conferences and workshops, in addition to these main international events, at the regional and national levels.

13.2.2 Hypermedia Systems: An Example

Actually, it is not easy to present on paper a real hypermedia system. Therefore, it is urgently recommended to the reader to work with such a system (e.g., the NCSA Mosaic© tool for viewing hypermedia documents written in HTML – an application of SGML) to get a better understanding of the properties and the advantages and disadvantages.

The following example of a *lecture "hypertext"*, as a hypermedia document, is similar to [Nie90b, Nie90a]. It describes the part of a typical manipulation process with a hypermedia system.

First Screen

A possibly natural environment is created for the reader of a hypertext to improvise the usual setup (Figure 13.8). Therefore, at the beginning of the lecture, a book is shown, which can be opened by clicking on it. The title of the document appears on the cover.

Additional information can also be shown, such as when the reader last opened the book. The same information (after the book was opened) is only shown with respect to the individual nodes.

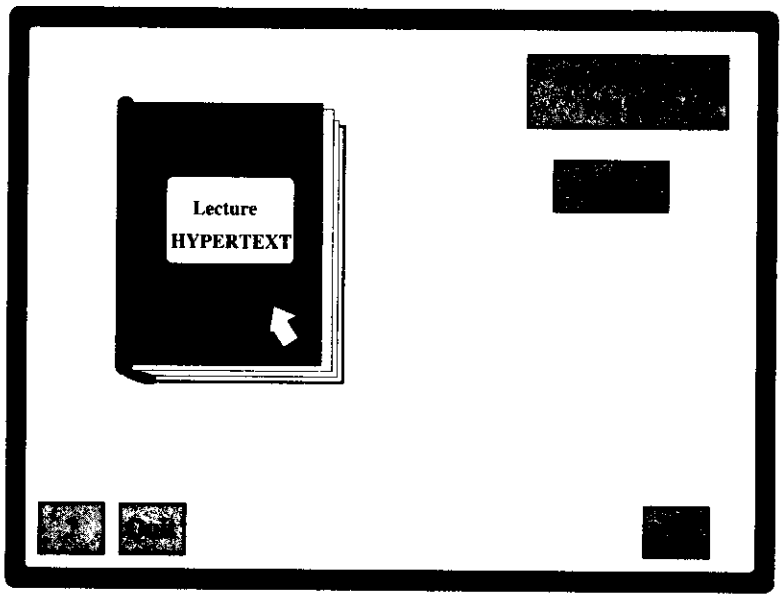

Figure 13.8: *The first screen of a hypermedia lecture.*

A content-sensitive help function (with respect to the particular state of the system) can be made available. The help function describes the actual state and the implications of possible interactions with the user. Further, it allows for the possibility to access general help information. An application of hypertext, which is very popular, consists of such system-wide help functions. Here, the text medium is most often used.

Second Screen

Upon opening the book (Figure 13.9), the reader is presented with an overview of the document in the form of a two-dimensional content directory. There is no first chapter. Besides the linking nodes with edges (i.e., the information units with pointers), the physical ordering of the nodes among each other can provide additional information. Hence, chapters that are closely related to each other can be presented on the screen near each other. The author can express semantic relations through the layout. For example, if a document describes ODA, SGML and hypertext,

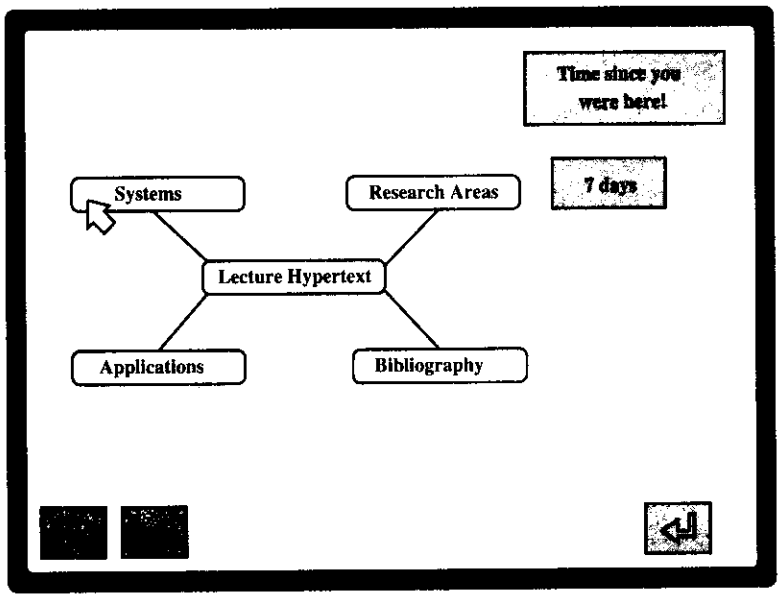

Figure 13.9: *The second screen – first-level content directory.*

(1) ODA and SGML can be placed close together during the presentation of both document architectures, and (2) *hypertext* can be used in both architectures.

The content directory in its most general form is often represented by a tree graph. The nodes that have already been visited can be marked as such, which simplifies navigation. When clicking at a node, only a paragraph is shown because of overview reasons.

On one hand, if too many nodes are linked in the content directory, the reader loses the comprehensive view. Given a 14" screen with VGA resolution and a content directory in form of a tree, a practical number of displayed nodes is 30. It is assumed that all nodes are simultaneously presented. On the other hand, if too few nodes are shown (e.g., only three), then this may lead to a large number of levels which will cause a fragmented perspective. The optimal number of presented nodes depends on the screen size, the number of pointers and the degree of possible levels. Many nodes can be presented as trees, few nodes can be part of complex graphs. Mostly, content directories are trees. The content of the particular destination node refines the content directory. In our example (Figure 13.9), *Lecture Hypertext* is divided

into four sections: *Applications*, *Systems*, *Research Areas* and *Bibliography*. The reader selects *Systems* here.

Third Screen

The structure of the four sections on the third screen can be presented in a refined version. The four sections can be seen as *buttons* which the user can select.

The following *Systems* subtopics are offered: *Requirements*, *User Interface Design* and a *Classification* (Figure 13.10). In the presentation (Figure 13.10), a multilevel

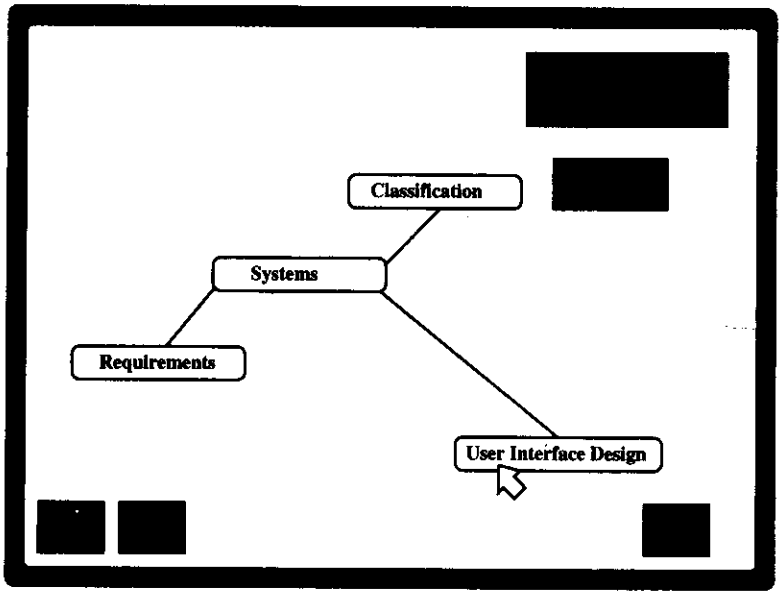

Figure 13.10: *The third screen – second-level content directory.*

content directory could be hidden, but we assume this is not the case. The user may get information about the user interface and select this information unit. Here, the reader selects *User Interface Design.*

Fourth Screen

Typical information about the medium text, used in *User Interface Design*, is presented on the fourth screen (Figure 13.11). In addition, the book's content directory

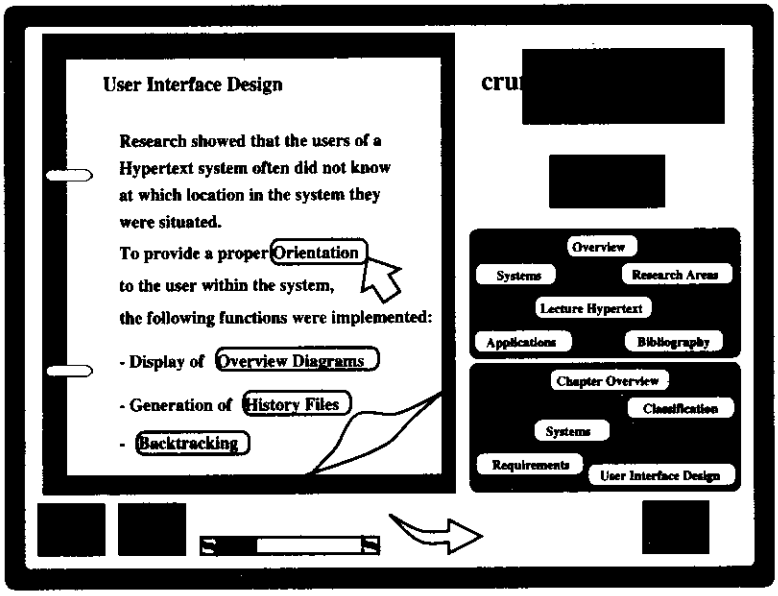

Figure 13.11: *The fourth screen – details about User Interface Design.*

and the second level are visible. The selected path *Systems/User Interface Design* is specially marked. This helps the reader to navigate through the document. Text is displayed on one side of the screen imitating an opened book. Some parts of the screen are marked as anchors: *Orientation, Overview Diagrams, History Files* and *Backtracking*. Each of these anchors points to further information. This can lead to faster pacing without absorbing previous information. Further information about the topic *Orientation* is available as a motion video. The user clicks on the anchor *Orientation* (Figure 13.11) and the screen shown in Figure 13.12 appears.

Fifth Screen

The fifth screen (Figure 13.12) leads the reader to a video clip showing examples of *Orientation*. Here, there may also be an additional program for motion picture

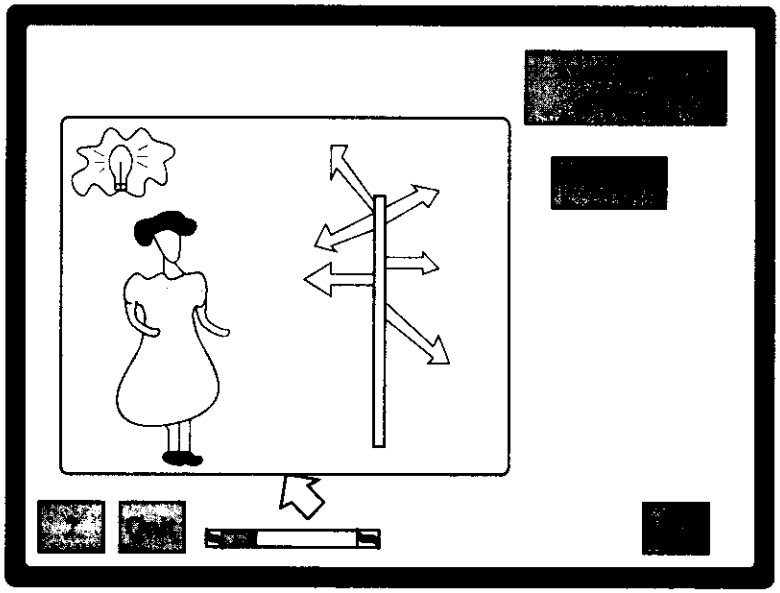

Figure 13.12: *The fifth screen - details about Orientation.*

control. This program provides functions similar to a video recorder, meaning the video can be moved forward and backward at different speeds, as well as stopped, and a still image can be displayed. Additionally, certain positions in the video clip (e.g., the beginning of the clip) should be made accessible.

The selection of the symbol at the lower right corner leads the reader back to the preceding node, in this case, to the *User Interface Design* screen (Figure 13.13).

Sixth Screen

The sixth screen is a return to the User Interface Design screen. The user now selects History Files. The cross-bar under the opened book is a scroll-bar. It indicates that the displayed page represents only approximately 25% of all the information stored

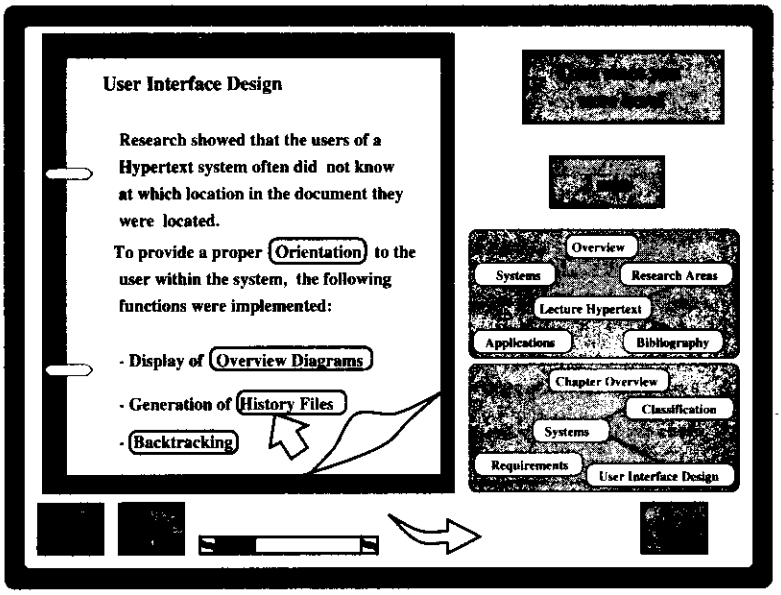

Figure 13.13: *Hypermedia example: sixth screen – details about the user interface.*

in the node about *User Interface Design*. Additional pages can be viewed by rolling the roll-bar or selection of the pointer.

Seventh Screen

The *History Files* screen (Figure 13.14) shows the particular nodes last read. Specifically, the reader read the *cover page* 5 minutes ago, the overview about the *lecture Hypertext* 2 minutes ago, the chapter *Systems* 4 minutes ago, and the *User Interface Design* in the chapter *systems* 2 minutes ago. This history list shows all of the traversed nodes. Each node can be accessed again by selecting an element in the list.

Often, a graphical assignment is better to keep in mind than a textual description. The user, for example, may remember that there is a picture where a red hat is placed in the upper left corner. Alternatively, a display of the traversed nodes may help the reader to find his/her way through the document. This is not a very good solution, however if many screens have been involved.

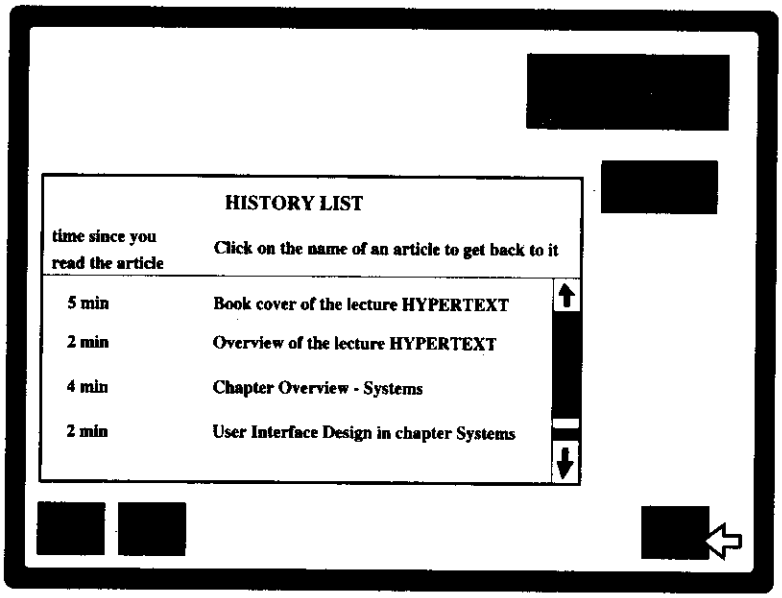

Figure 13.14: *The Seventh screen – history of previously presented information.*

With this short, virtual navigation through a hypertext lecture, some characteristic properties of a hypertext system have been shown. Various systems use different concepts, functions and layout of the user interface.

Further Application Areas

A lecture is not the only respectively typical application area. The following areas have shown to be useful domains for hypertext systems:

- In some classical computer applications, hypertext is already state-of-the-art; especially, the help function.
- In the area of commercial applications, repair and operational manuals can be found. Here, different media are used. This technology leads to a replacement of a microfilm which can be found by the distribution of replacement parts. A repair instruction can be presented much more flexibly using motion pictures. Here, an interaction in the form of a slow forward and backward rewinding

is expected. Exhibition and product catalogues create, together with other applications from the advertisement branch, the basis for a number of diverse applications.

- The organization of ideas, brainstorming and the generation of scientific documents count, for example, as intellectual applications. Here, the structure of the document is not clear at the beginning of its generation. During the intellectual process of writing a document, the structure gets clarified.

- Education and tutorials can be improved through the input of continuous media. Foreign language education requires the audio medium. In museums, further explanation of exhibition pieces can be offered to visitors using audio and video.

- Tourist information systems and interactive science-fiction movies count on the areas of entertainment and free-time activities. A new generation of computer games is going to become available.

Hypertext provides advantages whenever the document for a specific application does not require strict linear structure. However, the authors' experiences show that hypertext will not replace all conventional print material.

13.2.3 History

The history of hypertext goes quite far back, although it has been only recently (in the last couple of years) that hypertext systems came on the market. Also, the integration of continuous media was demonstrated in laboratories several years ago. In the following paragraph, we will give a short overview of hypermedia/hypertext history according to [Nie90b].

Vannever Bush is the originator of the main hypertext concept, the linked information structure. He described the first hypertext system *Memex* (MEMory EXtender). Memex was never implemented, it exists only on the paper. Vannever Bush developed the ideas for this topic in 1932. He published the first descriptive article as *We May Think* in 1945.

Let us imagine all information in the form of microfilm being placed on a table. With proper projectors, areas could be displayed in similar form as using X Window System™. Here, an *associative index* is generated which links the different microfilm areas together (these are the pointers).

Doug Englebart developed a project to augment the human capability *Augment* at the Stanford Research Institute (SRI) 1962-1976. One part of it is *NLS (oN Line System)*, which has hypertext properties. NLS served as joint document storage for all created documents during this project. All scientists working on this project used it with its possibilities of pointers. At the end, there were approximately 100,000 entries.

Ted Nelson used the notion *Hypertext* for the first time in 1965. In his system *Xanadu*, all information which human beings described at any time, was contained. His concepts described the access to local, as well as to remote data. Xanadu was not implemented with his global information content until now.

Since the middle of the 1960s, work on hypertext systems has been going on at Brown University, Providence, RI. In 1967, the *Hypertext Editing System* was developed under the leadership of Andries van Dam. This was the first run-able hypertext system. It needed 120-Kbyte main memory of a small IBM/360 mainframe computer. It was sold and used for the documentation of the Apollo mission. The successor project was *FRESS (File Retrieval and Editing SyStem)* in 1968. Both systems linked documents through pointers, the user interface was implemented through text. At Brown University from this time, successful research in the area of hypertext/hypermedia has continued.

The *Aspen Movie Map* is probably the first important hypermedia system which supports continuous media too. It was developed at the *MIT Architecture Machine Group* under the intensive cooperation of Andrew Lippman. This group was built up later on with other scientists and was known as the *MIT Media Lab* [Bra87]. With this application, a virtual drive through the city Aspen (Colorado) could be followed on the computer screen. The user could move in all four geographical directions as s/he desired. A *joystick* served as an input of the direction. The technique consisted of a large set of individual images which were stored on a video disk. For this purpose, four cameras were installed on a pick-up with the angle of

90° to each other (with the view: front, back, right, left). The car drove through all streets of the city and took one image every three meters. The images were linked through implicit pointers: therefore, a drive through the city could be simulated. The drive was simulated with a maximal two images per second; therefore, a speed of approximately 110 km/h could be achieved. The display occurred through two screens: the first screen displayed the picture of the street and the second screen showed a street map with the actual position.

Successor projects concentrated on the joint usage of video, individual images and text for bike and car repair-manuals.

Until now, all hypertext systems mentioned were not developed as products and seldom were used outside of their research groups. In 1982, Symbolics started development of the *Symbolics Document Examiner*. It was ready as the first hypertext product in 1985. Its main application was the documentation of the Symbolics Workstation, which was comprised of about 8,000 pages. It contained approximately 10,000 nodes and 23,000 pointers. Also, the metaphor *a book on the screen* was used and an emphasis was put on a simple user interface.

Since 1985, many hypertext systems have been announced and established on the market. *NoteCards* from Xerox and *Intermedia* from Brown University started as research projects and ended up as products. The *Guide*, implemented by Office Workstation Limited, started as product development. It was the first product based on mini computers (1986). In 1987, the Apple company presented the *HyperCard*. It was installed for free on all Macintosh computers and was therefore widely available.

Concepts

Hypertext systems differ from each other in their fundamental concepts:

- *Unspecific systems* were not developed for any specific application. They are determined to be used generally for the generation and reading of hypertext documents. The Apple (HyperCard) product is an example.
- *Application-specific systems* were developed for determined usage. For example, gIBIS gives explanations for political discussions. It is meant to be a

decision help. In gIBIS, three special nodes and nine different pointer types exist [CB88].

13.2.4 Systems: Architecture, Nodes and Pointers

Architecture

The architecture of a hypertext system can be divided into three layers with different functionalities [CG87]:

- *Presentation Layer*

  At the upper layer, the *presentation layer*, all functions connected to the user interface are embedded. Here, nodes and pointers are mapped to the user interface. At the user interface, one or several parts of the document are visualized. This layer determines, based on the given structure and user's desired display, which data are presented and how they are presented. This layer takes over control of all inputs.

- *Hypertext Abstract Machine*

  The *Hypertext Abstract Machine (HAM)* is placed between the presentation and storage layers. It can expect from the underlying layer database functions for storage of multimedia data in a distributed environment. It does not have to consider input and output of the upper layer. HAM knows the structure of the document, it has the knowledge about the pointers and its attributes. The data structure, respectively a document architecture, is constructed for the management of the document. This layer has the least system dependency in comparison to the other two layers. Therefore, it is the most suitable layer for standardization [Nie90b].

- *Storage Layer*

  The *storage layer* (also called the database layer) is the lowest layer. All functions connected with the storage of data, i.e., secondary storage management, belong to this layer. The specific properties of the different discrete and continuous media need to be considered. Functionalities from traditional

database systems are expected, such as persistence (data persist through programs and processes), multi-user operations (synchronization, locks, ...) and the restoration of data after a failure (transaction). The nodes and pointers of a hypertext document are processed as data objects without any special semantics.

Unfortunately, in most current implementation, there is no clear division between the different layers. The reasons are: shorter development time, efficient implementations and currently an incomplete, respectively unavailable general multimedia interface for the lowest layer.

Nodes

A *node* is an information unit (LDU) in a hypertext document. The main classification criterion of different realizations is the maximal stored data amount in one node:

- The maximal stored *data amount can be limited* and mapped onto the screen size. The metaphor of a note card, respectively a frame, was introduced here (e.g., HyperCard). A video clip and audio passage could be limited to the duration of, for example, 20 seconds.

  An author is forced eventually to distribute logical connected text content to several cards, although it is not desired. Applying it to video clips and audio passages, it would mean that the close interconnection among the distributed sequences could get lost easily. An advantage is the overview.

- Window-based systems with an *unlimited data amount* per node are the alternative. Forward and backward scrolling of pages is offered analogous to other windows at the user interface. *Intermedia* is such a system. Here, at every node the amount of data, coded as continuous media, is not limited (in principle) with respect to its duration.

  Therefore, individual nodes can include a very different length, although at first they may appear equal. Two different methods at the user interface are used for the presentation of further information: Either it is switched between

the nodes, or scrolling is used in one node with the usual mechanisms known in window systems.

A secondary criterion applies to the *time point of information generation.* Usually, the author specifies the whole content of the nodes during the generation of the document. Alternatively, the author can generate information according to his/her previous input during the presentation time. This approach allows the author to include, for example, information about a company, also a pointer, to the actual course of the company's stocks on the New York stock exchange. This information can be requested automatically through the *videotext service* and presented as part of the hypertext document.

Pointers

Pointers are the edges of a hypertext graph. Hypertext systems are classified according to different criteria with respect to edges. As a first question, one can ask: *Which information includes a pointer?*

- *Simple pointers* link two nodes of the graph without containing any further information. They are visible only through the relation between the nodes.
- *Typed pointers* include, in addition to the link between two nodes, further information. Each pointer gets a *label.* Through this label, commentaries to the particular label are possible (e.g., author and creation date).

  One can use further semantics. For example, in the case of an educational unit, the continuation of reading further details could depend on the result of an exam testing the previous details. The pointers then include a formula to activate the further reading. The formula is dependent on the result of the test. Also, access rights can be controlled through pointers. Another possibility consists of assigning types to pointers according to their properties. For example, it can be used to differentiate between the relations *destination node is an example* and *destination node is a detail.* These different semantics can also be expressed through different representations of the anchor at the user interface.

Another property of the pointers is connected to the question: *What does the pointer mean?* Often, pointers with very different meanings are used together. This usage complicates the understanding. The author of a hypertext should know about this problem and use unambiguous pointers. The following relations can be expressed through pointers:

- *To be*: A is part of B. This sentence represents a set relation.
- *To present*: A is an example of B, A demonstrated B.
- *To influence*: A causes B, B is a result of A. Consequences from a behavior can be described more closely.
- *To need* or *to be needed*: A needs B, B is needed by A. This relation expresses a necessity.
- *To own*: A has B, A is associated with B. Here, ownership is expressed.
- *To include*: A includes B, A consists of B, A occurs in B. An inclusion relation is expressed in different meanings.
- *To be similar*: A is similar to B, A is different from B, A replaces B, A is the alternative to B. Using this relation, similarities can be expressed.

Another basic property of pointers is described with the question: *Who is responsible for the pointer specification?* There are two possibilities:

- *Implicit Pointers*

  A relation between nodes can be established automatically by a hypertext system. The author determines the algorithm according to which pointers are created. The system *Intermedia* automatically generates all pointers which belong to one index. A similar approach can be taken by lexicons. Query references are done automatically using main notions of an entry.

- *Explicit Pointers*

  The author creates all links by him/herself.

A pointer can be created at different times. The question follows: *When is the destination of a pointer specified?*

- In the classical case, the pointer is created during the generation of a hypertext document, and hereby the origin of the destination node is determined. The author determines explicitly the links of the information units during document processing.

- A destination node can be determined first by using the pointer, i.e., during reading. The author specifies an algorithm for the creation of the pointers, but they are determined first during the reading depending on the context. The system computes the destination node.

  An example is a travel plan which shows the trains to one specific destination station. A pointer to the next train depends on actual time. This example comes from the information system to the city Glasgow *Glasgow Online* [Har89].

In the most systems, a pointer has one source and one destination node. One can also ask the questions: *Which direction has a pointer?* and *What is the number of outgoing pointers?*

The direction is mostly unidirectional. *Backtracking* is supported by the system itself. Hence, the path always leads back to the source. The alternative would be to have bidirectional pointers, but then the anchor, as well as the destination node, would have to be specially marked. Introducing bidirectional pointers, it is possible to have multiple references from several nodes to the same destination node. Hence, these kinds of pointers must be marked at the destination node and a further choice criterion must exist. Most systems support unidirectional pointers with only one destination node. It is easier to understand and generate.

The last question is connected to the form of an anchor at the user interface: *How is the pointer represented?* Section 13.2.1 shows different possibilities.

Tools

A hypertext system consists of several necessary tools. *Editor(s)* process information represented in different media. Beside this, the generation, management, editing and deletion of pointers are supported.

Search tools allow the search of desired information. Also, different media need to be considered.

Browser allows a shortened but clear representation of the nodes and edges. The nodes are described media-dependently. The structure is presented to the user mostly in a graphical representation. Often, only the previously read text or relevant information can be displayed.

During the *navigation* through a document, a proper support of the phenomena *Getting Lost in the Hyperspace* is needed. A *backtracking* and clear representation of the whole structure with respect to the actual position should be available.

13.2.5 Some Final Comments about Hypertext Systems

The *ordering* during a reading of a hypertext document should be pre-specified with respect to the context and reader's interest. Therefore, the structure of the document can change depending on the context [Hof91].

For example, a textbook about the esophagus can offer to a medical student in his/her first semester an overview of the organ's functionality. Only a limited set of nodes are presented. The most pointers build a tree with a suggested path. A navigation through this document turns out to be simple. In the following semesters, the students are lead to specific topics in the surgery area. Using text, graphics, video and audio, different possibilities of surgical procedures can be presented. The hypertext document includes in this context, in addition to the fundamental notions, a detailed description of the surgery. Further sections in this document discuss research aspects. References point to a number of other articles, and also current unsolved research aspects are shown. The user has the possibility of navigating through the whole document.

The most current hypertext systems do not allow any references to nodes outside of their internal data structure. For example, shared electronic letters cannot be stored as documents in one hypertext system, and simultaneously sent through another mailing program. *Open solutions* would be required in this case to support links between information units of different hypertext applications. This requires standardized exchange formats and protocols.

A hypertext document is stored mostly on one computer. In a *distributed environment*, information units can be located on different computers. The pointers go beyond the computer boundaries. According to the presented architecture, the storage layer would be mostly affected.

Further interesting development and research projects refer to the following aspects and questions:

- Size and concept of the information units.

  (What size is the *optimal size*? Which factors does this specification depend on?)

- Support of distributed documents by migration of information and/or the reorganization of networks.

  (How can a document be preserved with respect to its remotely stored content?)

- Version management.

  (Which elements should comply with version management? What should this management look like?)

- Authorization and access rights.

  (Which elements should be subject to an authorization and what should their access rights be?)

- Cooperative work, joint document processing.

  (Which accesses should be locked? How is such a management implemented?)

- Virtual views onto hypermedia documents.

(What does the virtual view determine? How are the virtual views managed?)

Concluding, a short evaluation of the hypertext/hypermedia properties is given. A part of this evaluation is based on personal experiences of the authors, further ideas come from discussion with experts. Many aspects, which are visible at the beginning of working with such systems, are mostly positive. Operation of most systems can be learned quite easily without manual. The user knows quickly and effectively how to find the desired information and how to manipulate all data.

Many of these enumerated properties are system-dependent, but most hypertext systems show the named properties. The hypertext documents themselves are very different in nature. Some are structured clearly and easy to read, others are not structured properly, hence they are difficult to read.

Hypermedia integrates diverse media in a very elegant and simple fashion. Each relation between information units is implemented with pointers. Some systems also support the joint management of information among several persons. This technique has some properties which need to be critically evaluated. The most well-known effect is *Getting Lost in the Hyperspace.* While reading such a document, the perspective and context can get lost. Hypertext documents can easily become so called *spaghetti books* because of many pointers with different meanings. A hypertext document is difficult to bring into paper form without any information loss. Hence, even without audio and video information, the reader needs a computer. Some systems use their own window systems. There is a lack of established standards for information exchange among current hypertext systems.

Different task forces work on standards for hypermedia. Extensions of document architectures ODA and SGML include hypertext techniques. They support a data exchange of hypertext-like documents in a heterogeneous environment.

Further activities are embedded in ISO/IEC JTC1 SC2/WG12 *Multimedia and Hypermedia Information Coding Expert Group (MHEG).* This group works on the *coded representation of multimedia and hypermedia information* [MHE93].

The ANSI group X2V1.8M represents the *Music Information Processing Standards (MIPS) Committee.* This committee works on *HyTime* with respect to hypertext and the *Standard Music Description Language (SMDL).*

In summary, hypertext is not suitable for all kinds of documents and applications. This technique is very good with lexicons. The information units can be linked to each other through references (pointers). In comparison to an access and search in a book, the usage of hypertext system makes finding information faster.

Video and audio can be included in such a lexicon easily. For example, passages from music or animal voices can be stored under a certain entry's explanation. Video allows a short representation of typical movement processes as part of a lexicon entry.

Another area of using hypermedia is education and tutorials. Coursework may be supported by audio-visual means. These means can be used according to the learning behavior of the individual participants. For the instructors, the hypertext research works toward a didactical support for course design. For example, such a system was developed at the University of Karlsruhe [Mue89].

13.3 Document Architecture SGML

The *Standard Generalized Markup Language (SGML)* [Gol91b], [Org86] was supported mostly by American publishers. Authors prepare the text, i.e., the content. They specify in a uniform way the title, tables, etc., without a description of the actual representation (e.g., script type and line distance). The publisher specifies the resulting layout.

The basic idea is that the author uses *tags* for marking certain text parts. SGML determines the form of *tags*, but it does not specify their location or meaning. User groups agree on the meaning of the tags. SGML makes a *frame* available with which the user specifies the syntax description in an object-specific system. Here, classes and objects, hierarchies of classes and objects, inheritance and the link to methods (processing instructions) can be used by the specification. SGML specifies the syntax, but not the semantics. For example,

```
<title>Multimedia-Systems</title>

<author>Felix Gatou</author>
```

```
<side>IBM</side>

<summary>This exceptional paper from Peter ...
```

This example shows an application of SGML in a text document.

13.3.1 Some Details

Figure 13.15 shows the processing of an SGML document. It is divided into two

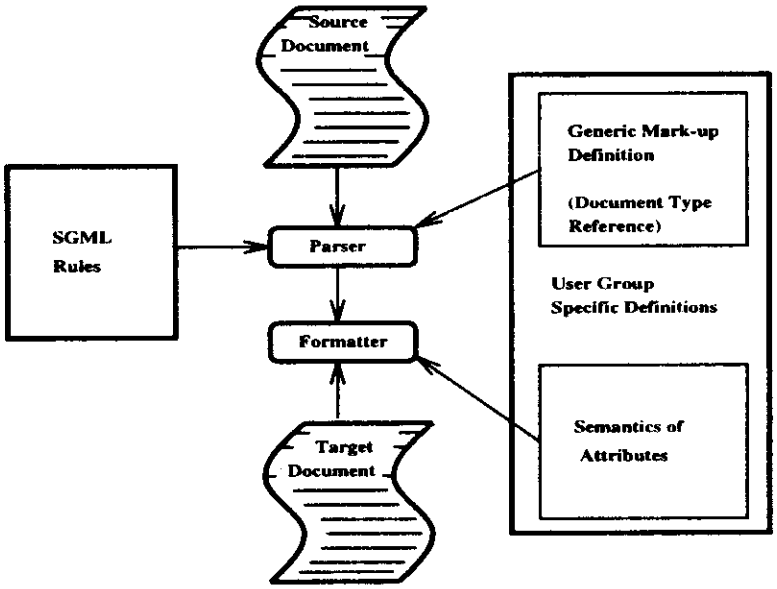

Figure 13.15: *SGML: Document processing – from the information to the presentation.*

processes. Only the *formatter* knows the meaning of the tag and it transforms the document into a formatted document. The parser uses the tags, occurring in the document, in combination with the corresponding document type. Specification of the document structure is done with tags. Here, parts of the layout are linked together. This is based on the joint context between the originator of the document and the formatter process. It is not defined through SGML.

Tags are divided into different categories:

- The *descriptive markup (tags)* describes the actual structure always in the form:

  `<start-tag> respectively also </end-tag>`

  An example is the definition of a paragraph at its beginning:

  `<paragraph> The text of the paragraph follows ..`

- The *entity reference* provides connection to another element. This element replaces the entity reference. This can be understood also as an abbreviation to which the actual content can be copied later at the corresponding place. The following example shows entity reference in a mathematical context:

  `&square x ....` should be x^2

- The *markup declarations* define the elements to which an entity reference refers. In our example of squaring a variable x, square is defined as:

  `<!ELEMENT square (...)>`

  A markup declaration can be used to define rules for the structure (the classes). The following example illustrates the construction of an article *paper*:

  `<!ELEMENT paper (preamble, body, postamble)>`

  `<!ELEMENT preamble (title, author, side)>`

```
<!ELEMENT title (#CDATA)> -- character data

<!ELEMENT body (...)>
```

- Instructions for other programs in a text are entered through *processing instructions.* They can be meant, for example, for the formatter. Using processing instructions, different media can be inserted.

SGML defines a syntax for tags through a grammar which needs to be followed. SGML does not define the semantics of these tags.

The *information or document architecture* of SGML is shown in Figure 13.16. SGML

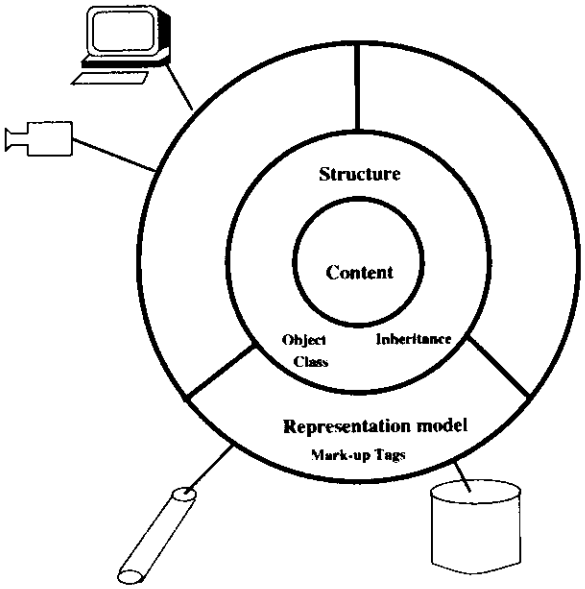

Figure 13.16: *SGML: Document architecture - emphasis on the representation model.*

with its tags possesses a representation model. Objects, classes and inheritance can be used for the definition of the structure.

13.3.2 SGML and Multimedia

Multimedia data are supported in the SGML standard only in the form of graphics. A graphical image as a CGM (*Computer Graphics Metafile*) is embedded in an SGML document. The standard does not refer to other media [Org86].

```
<!ATTLIST video id ID #IMPLIED>

<!ATTLIST video synch synch #IMPLIED>

<!ELEMENT video (audio, movpic)>

<!ELEMENT audio (#NDATA)> -- non-text media

<!ELEMENT movpic (#NDATA)> -- non-text media

...

<!ELEMENT story (preamble, body, postamble)> :
```

A link to concrete data can be specified through #NDATA. The data are stored mostly externally in a separate file.

The above example shows the definition of video which consists of audio and motion pictures.

Multimedia information units must be presented properly. The synchronization between the components is very important here. HyTime [Gol91a] and MHEG [MHE93] work on these issues.

13.3.3 Closing Comments about SGML

A standardized document exchange is necessary with respect to the communication. Sender (writer) and receiver (reader) can be distributed in time, as well as in space. Often, documents are processed automatically. This requires a joint context. The syntax is transmitted and the semantics must be discussed in SGML separately. The *Document Type Definitions* form the basis for these discussion.

SGML as a standard will stay in its current form [Org86], but extensions(additions) are also worked out. A standardized layout semantics is necessary. This will simplify interactions of user groups. The *Document Style Semantics and Specification Language (DSSSL)* is such an extension to the standard. Based on PostScript, a *Standard Page Description Language (SPDL)* is specified.

With respect to multimedia, pointers are included as non-readable. An extension for the description of music represents *Standard Music Description language(SMDL)* and HyTime.

Another application conforming to International Standard ISO 8879 – SGML – is the *HyperText Markup Language (HTML)*. HTML is a mark-up language for hypertext which is understood by all WWW (World Wide Web) clients. HTML is persuaded to become a standard for interchange of hypertext information on the network. It is proposed to be registered as a MIME (RFC1521) content type. HTML can be used to represent:

- Hypertext news, mail, online documentation and collaborative hypermedia.
- Menus of options.
- Database query results.
- Simple structured documents with in-lined graphics.
- Hypertext views of existing bodies of information.

13.4 Document Architecture ODA

The *Open Document Architecture (ODA)* [Org89] was initially called the *Office Document Architecture* because it supports mostly office-oriented applications. The main goal of this document architecture is to support the exchange, processing and presentation of documents in open systems. ODA has been endorsed mainly by the computer industry, especially in Europe.

13.4.1 Some Details on ODA

The main property of ODA is the distinction among content, logical structure and layout structure. This is in contrast to SGML where only a logical structure and the contents are defined. ODA also defines semantics. Figure 13.17 shows these three aspects linked to a document. One can imagine these aspects as three orthogonal

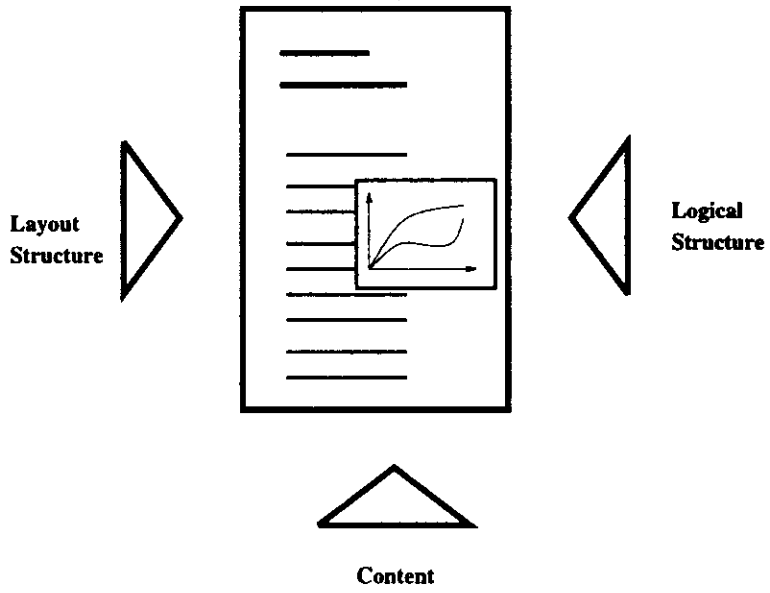

Figure 13.17: *ODA: Content, layout and logical view.*

views of the same document. Each of these views represent one aspect, together we get the actual document.

Content Portions

The content of the document consists of *Content Portions*. These can be manipulated according to the corresponding medium.

A *content architecture* describes for each medium: (1) the specification of the elements, (2) the possible access functions and, (3) the data coding. Individual elements are the Logical Data Units (LDUs), which are determined for each medium. The access functions serve for the manipulation of individual elements. The coding of the data determines the mapping with respect to bits and bytes.

ODA has content architectures for media *text, geometrical graphics* and *raster graphics*. Contents of the medium *text* are defined through the *Character Content Architecture*. The *Geometric Graphics Content Architecture* allows a content description of still images. It also takes into account individual graphical objects. This is similar to CGM. Pixel-oriented still images are described through *Raster Graphics Content Architecture*. It can be a bitmap as well as a facsimile.

Layout Structure and Logical Structure

The structure and presentation models describe – according to the information architecture – the *cooperation* of information units. These kinds of meta information distinguish layout and logical structure.

The *layout structure* specifies mainly the representation of a document. It is related to a two dimensional representation with respect to a screen or paper. The presentation model is a tree. Figure 13.18 shows the content of a document together with the layout structure. Using *frames* the position and size of individual layout elements is established. For example, the page size and type style are also determined.

The *logical structure* includes the partitioning of the content as shown in Figure 13.19. Here, paragraphs and individual headings are specified according to the tree structure. Lists with their entries are defined (example) as:

```
paper = preamble body postamble
```

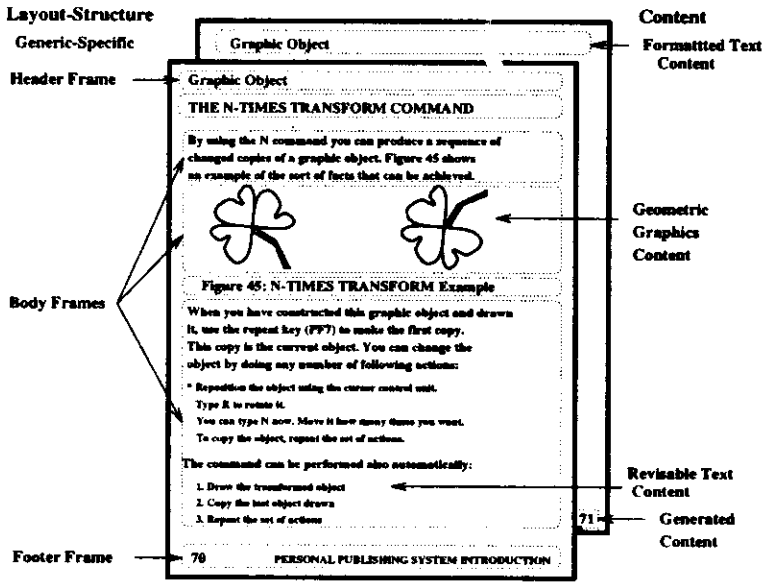

Figure 13.18: *Layout view and content of an ODA document.*

```
body = chapter1 chapter2

chapter1 = heading paragraph ...  picture ...

chapter2 = heading paragraph picture paragraph
```

The above example describes the logical structure of an article. Each article consists of a *preamble*, a *body* and a *postamble*. The body includes two chapters, both of them start with headings. A content is assigned to each element of this logical structure.

An example, shown in Figure 13.20, demonstrates the relation among a content, logical structure and layout structure. Figure 13.21 shows the contents and the logical and layout structures of the document from Figure 13.20. This figure consists of a title, a describing text and a figure. The title and describing text are mapped onto the content architecture *Character Content Architecture*. The figure belongs to the content architecture *Raster Graphics Content Architecture*.

The logical structure of the example dictates for each section at least one title, one paragraph and one figure. This behavior is shown in Figure 13.21.

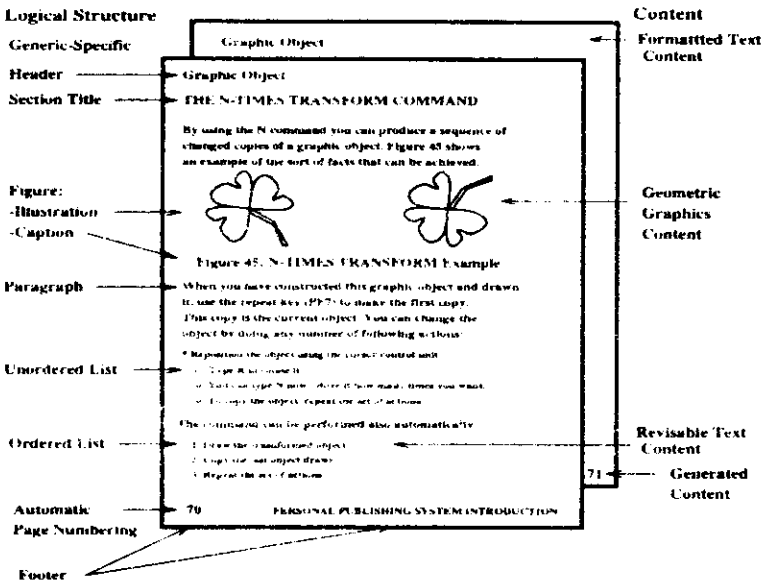

Figure 13.19: *Logical view and content of an ODA document.*

In the upper right paragraph of Figure 13.21, the layout structure of the document is displayed. It consists of several *frames* which are ordered in a certain style on the two-dimensional surface.

The ordering is presented through individual lines between the elements of the different areas. A *content portion* is assigned to each leaf of the logical tree (a basic object) and of the layout tree.

ODA distinguishes the following layout and logical structures:

- The *generic logical* and *generic layout structures* include a set of default values. For example, a paragraph can be specified with *LeftHandOffset = 0.*

- The *specific logical* and *specific layout structure* describe a concrete document. They are linked to the generic structure. For example, a concrete paragraph can be defined with *LeftHandOffset = 1cm.* The following example presents a specific layout object:

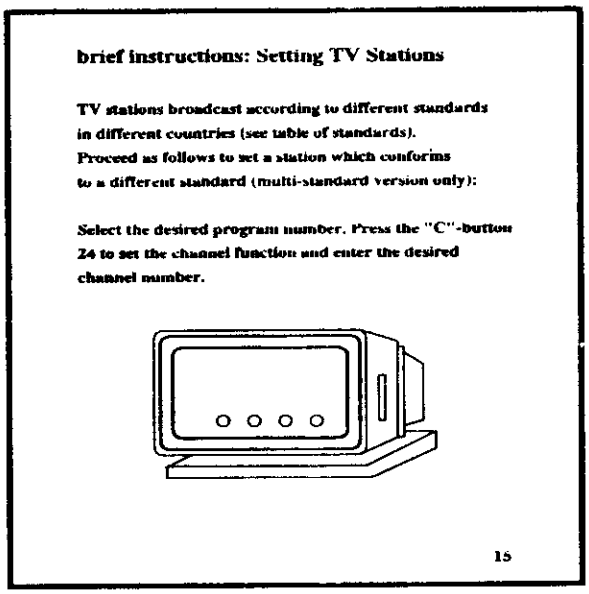

Figure 13.20: *An ODA document (example).*

object type:	block
object identifier:	title
position:	(x=111, y=222)
dimensions:	(height=333 width=444)
content architecture class:	formatted character content architecture
content portions:	(POINTER TO TEXT)

The information architecture ODA includes the cooperative models shown in Figure 13.22. The fundamental descriptive means of the structural and presentational models are linked to the individual nodes which build a document. The document is seen as a tree. Each node (also a document) is a *constituent*, or an object. It consists of a set of attributes, which represent the properties of the nodes. A node itself includes a concrete value or it defines relations between other nodes. Hereby, relations and operators, as shown in Table 13-1, are allowed.

If we consider document processing according to Figure 13.3, in ODA it can be presented as shown in Figure 13.23.

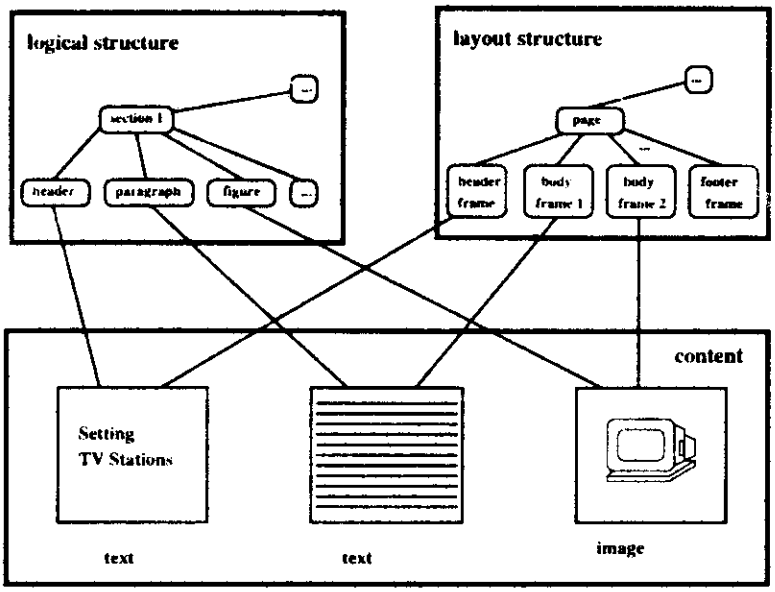

Figure 13.21: *Relations among content and logical and layout structures (example from Figure 13.20).*

The simplified distinction is between the editing, formatting (*Document Layout Process* and *Content Layout Process*) and actual presentation (*Imaging Process*). Current WYSIWYG (What You See Is What You Get) editors include these in one single step. It is important to mention that the processing assumes a linear reproduction. Therefore, this is only partially suitable as a document architecture for a hypertext system. Hence, work is occurring on *Hyper-ODA*. For the document exchange, different *document architecture classes* can be used, as shown in Figure 13.23:

- A *formatted document* includes the specific layout structure, and eventually the generic layout structure. It can be printed directly or displayed, but it cannot be changed.

- A *processable document* consists of the specific logical structure, eventually the generic logical structure, and later of the generic layout structure. The document cannot be printed directly or displayed. Change of content is possible.

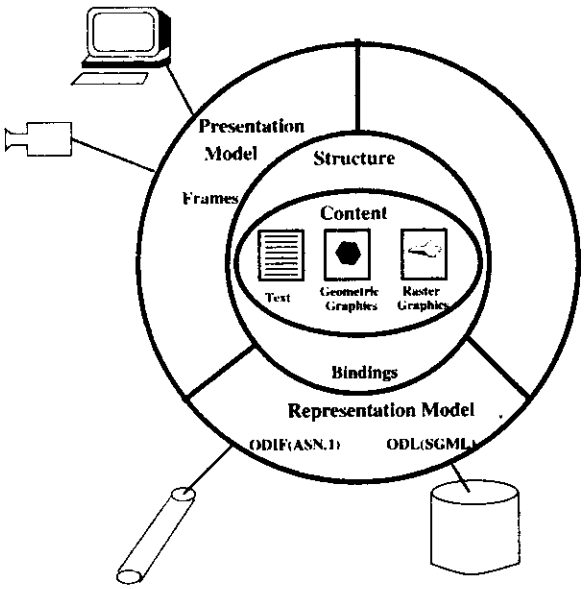

Figure 13.22: *ODA information architecture with structure, content, presentation and representation model.*

- A *formatted processable document* is a mixed form. It can be printed, displayed and the content can be changed.

For the communication of an ODA document, the representation model, shown in Figure 13.22, is used. This can be either the *Open Document Interchange Format (ODIF)* (based on ASN.1), or the *Open Document Language (ODL)* (based on SGML).

The *manipulation model* in ODA, shown in Figure 13.22, makes use of *Document Application Profiles (DAPs)*. These profiles are an ODA extension for document manipulation. Because of the ODA structure, the extension defines three different levels, which represent a subset of ODA (*Text Only, Text + Raster Graphics + Geometric Graphics, Advanced Level*).

Sequence	all child nodes are ordered sequentially
Aggregate	no ordering among the child nodes
Choice	one of the child nodes has a successor
Optional	one or no (operator)
Repeat	one .... any times (operator)
Optional Repeat	0 ... any times (operator)

Table 13.1: *Relations among nodes.*

13.4.2 ODA and Multimedia

Multimedia requires, besides spatial representational dimensions, the *time* as a main part of a document. If ODA should include continuous media, further extensions in the standard are necessary. Currently, multimedia is not part of the standard. All further paragraphs discuss only possible extensions, which formally may or may not be included in ODA in this form.

Contents

The *content portions* will change to *timed content portions* [RG90]. Hereby, the duration does not have to be specified *a priori*. These types of content portions are called *Open Timed Content Portions*. Let us consider an example of an animation which is generated during the presentation time depending on external events. The information, which can be included during the presentation time, is images taken from the camera. In the case of a *Closed Timed Content Portion*, the duration is fixed. An example is a song.

Structure

Operations between objects must be extended with a time dimension where the time relation is specified in the father node v in proportion to the child nodes k_1, k_2. The following additional examples and described relations can be found in chapter 15 on

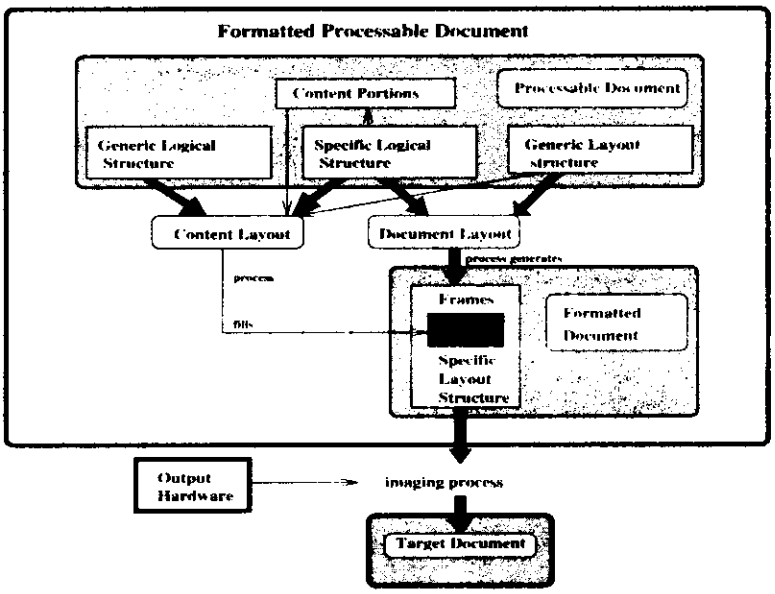

Figure 13.23: *ODA document processing - from the information to the presentation.*

synchronization:

- *before*: k_1 before k_2, $k_1 + k_2 > v$
- *meets*: $(k_1, k_2), k_1$ exactly before $k_2, k_1 + k_2 = v$
- *overlaps*
- *during*
- *start*
- *end*
- *middle*
- *at*
- *equal*

Content Architecture

Additional content architectures for audio and video must be defined. Hereby, the corresponding elements, LDUs, must be specified. For the access functions, a set of generally valid functions for the *control* of the media streams needs to be specified. Such functions are, for example, *Start* and *Stop*. Many functions are very often device-dependent. One of the most important aspects is a compatibility provision among different systems implementing ODA.

During data coding, a link to the existing *de-jure* and *de-facto* standards is required. Especially important standards are: JPEG for image compression, MPEG for video and audio, H.261 for video and CD technology. The possibility of an *open architecture* should be considered. With this approach, further developments could be considered today without changing or extending the standard.

Logical Structures

Extensions for multimedia of the logical structure also need to be considered. For example, a film can include a logical structure. It could be a tree with the following components:

1. Prelude
   - Introductory movie segment
   - Participating actors in the second movie segment
2. Scene 1
3. Scene 2
4. ...
5. Postlude

Such a structure would often be desirable for the user. This would allow one to deterministically skip some areas and to show or play other areas.

A time relation between different objects is also relevant for the definition of the logical structure. This time relation should be specified only to the extent that the user perceives content portions to be *in sync*. A relation can be defined between the subtitles and the scene as a whole, although the exact time cannot be specified by the logical structure. This would rather count the layout structure.

A spatial relation between different objects is defined in the same way as it is done by other discrete media. Instead of a tree structure in an ODA multimedia document, it should be possible to define a graph. This would allow for any kind of hypermedia techniques.

Layout Structure

The layout structure needs extensions for multimedia. The time relation by a motion picture and audio must be included. Further, questions such as *When will something be played?*, *From which point?* and *With which attributes and dependencies?* must be answered.

The spatial relation can specify, for example, relative and absolute positions by the audio object. Additionally, the volume and all other attributes and dependencies should be determined.

Especially by the continuous media, *interactivity* needs to be considered. The document is not only anymore a paper, the *linear* processing will become the *interactive* processing. In the case of ODA, the *imaging process* should not be left out, as we will discuss in the next section in terms of MHEG.

If all extensions of ODA with respect to the integration of continuous media are summarized, the result is the multimedia document architecture shown in Figure 13.2.

13.5 MHEG

The committee ISO/IEC JTC1/SC29 (*Coding of Audio, Picture, Multimedia and Hypermedia Information*) works on the standardization of the exchange format for multimedia systems. The actual standards are developed at the international level in three working groups cooperating with research and industry. Figure 13.24 shows that the three standards deal with the coding and compression of individual media. The results of the working groups: the *Joint Photographic Expert Group (JPEG)*

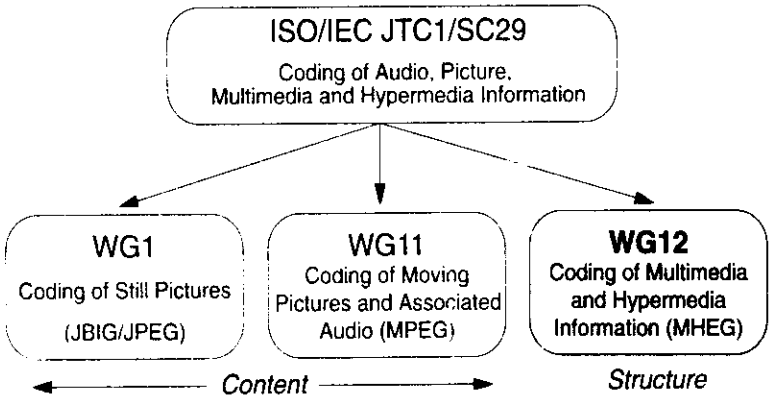

Figure 13.24: *Working Groups within the ISO-SG29.*

and the *Motion Picture Expert Group (MPEG)* are of special importance in the area of multimedia systems (see Chapter 6 on compression).

In a multimedia presentation, the contents, in the form of individual information objects, are described with the help of the above named standards. The structure (e.g., processing in time) is specified first through timely spatial relations between the information objects. The standard of this structure description is the subject of the working group WG12, which is known as the *Multimedia and Hypermedia Information Coding Expert Group (MHEG)*. The name of the developed standard is officially called *Information Technology - Coding of Multimedia and Hypermedia Information (MHEG)*. The final MHEG standard will be described in three documents. The first part will discuss the concepts, as well as the exchange format. The second part describes an alternative, semantically to the first part, isomorph syntax of the exchange format. The third part should present a reference architecture for

a linkage to the script languages. The main concepts are covered in the first document, and the last two documents are still in progress; therefore, we will focus on the first document with the basic concepts. Further discussions about MHEG are based mainly on the committee draft version, because: (1) all related experiences have been gained on this basis [ME94]; (2) the basic concepts between the final standard and this committee draft remain to be the same; and, (3) the finalization of this standard is still in progress while printing this book. Note that the following discussion is based on [ME94] designing, implementing and improving the MHEG standard.

13.5.1 Example of an Interactive Multimedia Presentation

Before a detailed description of the MHEG objects is given, we will briefly examine the individual elements of a presentation using a small scenario. Figure 13.25 presents a time diagram of an interactive multimedia presentation. The presentation starts with some music. As soon as the voice of a news-speaker is heard in the audio sequence, a graphic should appear on the screen for a couple of seconds. After the graphic disappears, the viewer carefully reads a text. After the text presentation ends, a *Stop button* appears on the screen. With this button the user can abort the audio sequence. Now, using a displayed input field, the user enters the title

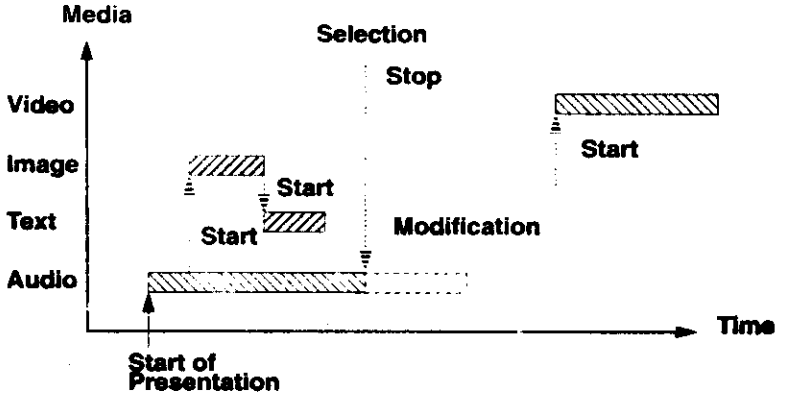

Figure 13.25: *Time diagram of an interactive presentation.*

of a desired video sequence. These video data are displayed immediately after the

modification.

Content

A presentation consists of a sequence of information representations. For the representation of this information, media with very different properties are available. Because of later *reuse*, it is useful to capture each information LDU as an individual object. The contents in our example are: the video sequence, the audio sequence, the graphics and the text.

Behavior

The notion *behavior* means all information which specifies the representation of the contents as well as defines the run of the presentation. The first part is controlled by the actions *start*, *set volume*, *set position*, etc. The last part is generated by the definition of timely, spatial and conditional links between individual elements. If the state of the content's presentation changes, then this may result in further commands on other objects (e.g., the deletion of the graphic causes the display of the text). Another possibility, how the behavior of a presentation can be determined, is when external programs or functions (script) are called.

User Interaction

In the discussed scenario, the running animation could be aborted by a corresponding user interaction. There can be two kinds of user interactions. The first one is the *simple selection*, which controls the run of the presentation through a prespecified choice (e.g., push the Stop button). The second kind is the more complex *modification*, which gives the user the possibility to enter data during the run of the presentation (e.g., editing of a data input field).

Container

Merging together several elements as discussed above, a presentation, which progresses in time, can be achieved. To be able to exchange this presentation between the involved systems, a *composite* element is necessary. This element is comparable to a container. It links together all the objects into a unit. With respect to hypertext/hypermedia documents, such containers can be ordered to a complex structure, if they are linked together through so-called hypertext pointers.

13.5.2 Derivation of a Class Hierarchy

Figure 13.26 summarizes the individual elements in the MHEG class hierarchy in the form of a tree. Instances can be created from all leaves (roman printed classes). All

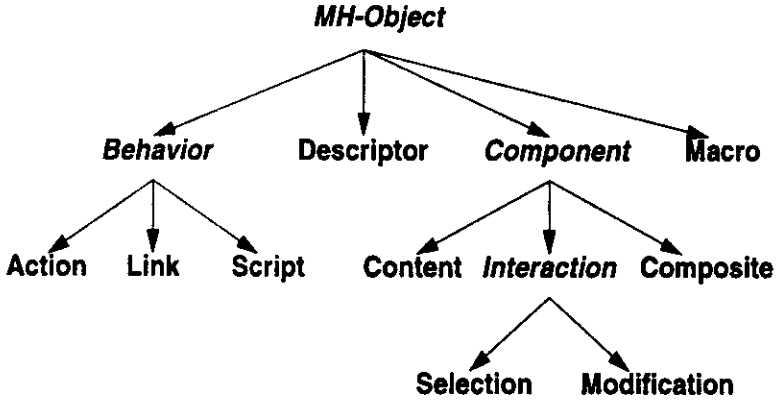

Figure 13.26: *Class hierarchy of MHEG objects.*

internal nodes, including the root (*italic* printed classes), are abstract classes, i.e., no instances can be generated from them. The leaves inherit some attributes from the root of the tree as an abstract basic class. The internal nodes do not include any further functions. Their task is to unify individual classes into meaningful groups. The action, the link and the script classes are grouped under the behavior class, which defines the behavior in a presentation. The interaction class includes the user interaction, which is again modeled through the selection and modification class. All the classes together with the content and composite classes specify the individual

components in the presentation and determine the component class. Some properties of the particular MHEG engine can be queried by the descriptor class. The macro class serves as the simplification of the access, respectively reuse of objects. Both classes play a minor role; therefore, they will not be discussed further.

The development of the MHEG standard uses the techniques of object-oriented design. Although a class hierarchy is considered a kernel of this technique, a closer look shows that the MHEG class hierarchy does not have the meaning it is often assigned. Methods of the classes are not specified in the standard, therefore only some attributes of the MH-object class are passed from their derived classes. In this context, a problem of using ASN.1 syntax for MHEG class description becomes obvious. ASN.1 does not support inheritance mechanisms; therefore, the reuse of the attributes in the remaining classes, which are defined in the MH object class, is applied. It needs to be mentioned that the MHEG class hierarchy is used mainly because of didactical reasons. For the implementation of a MHEG run-time environment, it does not have any importance.

MH-Object-Class

The abstract MH-Object-Class inherits both data structures *MHEG Identifier* and *Descriptor.*

MHEG Identifier consists of the attributes *MHEG Identifier* and *Object Number* and it serves as the addressing of MHEG objects. The first attribute identifies a specific application. The *Object Number* is a number which is defined only within the application. The data structure *Descriptor* provides the possibility to characterize more precisely each MHEG object through a number of optional attributes. For example, this can become meaningful if a presentation is decomposed into individual objects and the individual MHEG objects are stored in a database. Any author, supported by proper search functions, can reuse existing MHEG objects.

In the following paragraphs, the MHEG classes are presented and some main mechanisms are discussed. For more technical details, the interested reader should see the committee draft [MHE93].

13.5.3 Contents

Content Class

The content class differs from the other classes because it provides the link to the actual contents. Through this content class, this information becomes flexible and is linked together in an open way in the system. Each content object represents exactly one information within a presentation.

The type of the particular medium is defined in a content object through the attribute *MHEG Classification* and the coding is specified through the attribute *Hook*. The actual data can be included either in the object (*included data*), or they can be referenced through an unambiguous identifier (*referenced data*). The included data are meaningful only when a small amount of data is present. The reason is efficiency because the content object itself is transformed, before any exchange occurs, through the encoder/decoder. The referenced data have the advantage that they can be reused outside of MHEG.

At the time of content object processing, it must be guaranteed that the referenced data were requested through proper application services. For the presentation of data, one medium-corresponding representation component is used.

The standard includes a set of codings. Besides the existing standards, future standard (*Non-MHEG Standardized Catalogue*), as well as application-specific coding methods (*Proprietary Catalogue*), are considered. This is done through an open definition of the individual formats.

Virtual Coordination Systems

The so-called *Generic Space* defines a virtual coordination system. Content objects can be defined relative in dimension and ordering to each other. There are three axes: X (width), Y (height) and Z (depth). A value from -32768 to 32767 is assigned to each axis. During the run-time, a translation from the virtual MHEG coordinates to the physical coordinate system is performed in the particular presentation service (e.g., the number of pixels which cover a Motif window). Additionally, a time

coordinate system exists with its axis T. The defined value set for this axis is an interval from 0 to infinity where the scale unit is a millisecond.

Virtual Views

Until now we assumed that the presentation of the contents occurs exactly as originated. Actually, MHEG provides a set of possibilities which can control a presentation of the content objects through proper parameters. For example, a movie can be played according to certain time specifications, the volume of an audio sequence can be set or the visual area of a graphic can be specified. The manipulation of these parameters is determined through corresponding commands (see action class) in the coding. The following example of a computer simulated basketball game shows that the same player of each team can occur at different positions of the field at the same time. Instead of storing all possible presentation combinations as separate content objects, the change of the parameter is modeled as a call of an object's method. Using this approach, so-called virtual views (*Presentable*) to each content object are created during run-time. These virtual views are specified at the presentation-composition through unambiguous numbers. During run-time, they fix the parameters according to the representation. Figure 13.27 shows some examples of virtual views and illustrates the reuse possibilities.

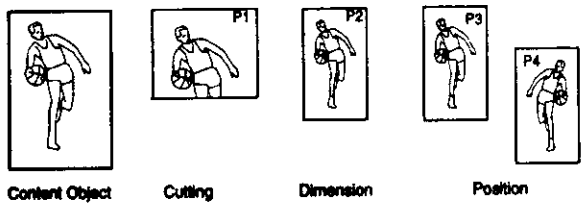

Figure 13.27: *Examples of virtual views.*

Such views exist for all component-classes, as well as for objects of the selection, modification and composite classes. However, they have within each class different specifications. In the next paragraphs, if MHEG objects are addressed, virtual views are meant. Only in the case of a different handling, the exclusion of virtual views will be stated explicitly.

13.5.4 Behavior

Action Class

The behavior of individual MHEG objects is determined with the *action class*. According to the object-oriented terminology, an action-object is a message which is sent to a MHEG object. By the destination object, a corresponding method is called which performs a change on this object. In an action object, the destination object is not specified. Hence, the action class represents only the interface (set of all public methods) to influence the presentation of individual MHEG objects. The implementation of the individual methods is hidden in the encapsulation of its specifics. Further, the capability of polymorphism is often used. This means that actions with the same name but with different specifications can exist. Altogether this leads to a very homogeneous interface (e.g., the display of objects occurs independently of the medium with the action *run*).

Individual actions can be classified into different groups depending on which method they influence. It is apparent that not each action for each MHEG object, respectively virtual view, is meaningful.

States and State Transitions

Some actions cause state changes on MHEG objects which are very important. The reader should recall the above described scenario where after the ending of the graphics display, a text was displayed. To specify such a relation between the presentation of the graphic and the text, a *state* needs to be defined where the virtual view of the graphics is captured (graphics is displayed or not displayed). For this purpose, the standard defines *state transitions* through protocol engines. Figure 13.28 shows two examples of state transition graphs. They are quite simple because of the low number of states. The *preparation status* represents the availability of an MHEG object. An example of a content object explains the status. At the beginning the object is in the *not ready* state. With the action *prepare*, referenced data are eventually localized and the necessary subsystem is initialized for the presentation. If these activities are performed successfully, the state changes to *ready* and the

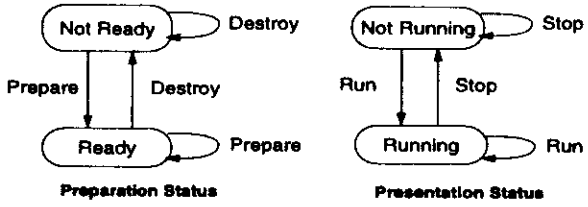

Figure 13.28: *An example of two state transition graphs.*

object can be displayed by a virtual view. With the action *destroy*, the resources, allocated by the object, can be freed. The display of this object depends on the *presentation status* of a corresponding virtual view.

Further state transitions are defined for the *composition status, modification status, time-stone status* and *script activation status.* An exception is the *selection status.* The different statuses are determined first with the modeling of the individual user interactions in a presentation. The previously described actions are basic operations of an MHEG object.

The construction of an action object also includes *complex runs* that are defined by grouping several basic actions. Depending on requirements, single actions can be performed in parallel or sequentially. The data structure of the action class defines for this purpose a *Parallel Action List.* The elements of this list consist again of a list of basic actions. *Delayed Sequential Action* is defined as a *Sequence of Actions.* The basic actions in *Delayed Sequential Action* are processed only sequentially. Figure 13.29 gives a graphical overview of the data structure.

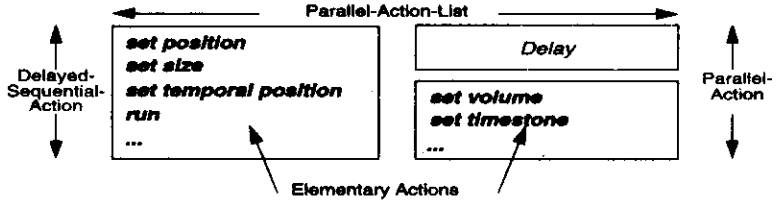

Figure 13.29: *An example of an action object.*

Link Class

So far, no relation which specifies the run of a presentation could be established between an action object and a content object, respectively a virtual view. The link class must fulfill two tasks. First, it specifies which actions are sent to which MHEG objects. Second, the conditions are specified under which this process occurs. From these tasks, it can be derived that the processing of an MHEG-coded presentation is based on an event-driven processing model. This model is suitable for the mapping of parallel running, synchronized processes which exist often in interactive multimedia presentations. Their sequentialization depends on the performing system.

A link object consists of a set of links. The semantics of a link are shown in Figure 13.30. A link connects a source object with one or several destination objects. The

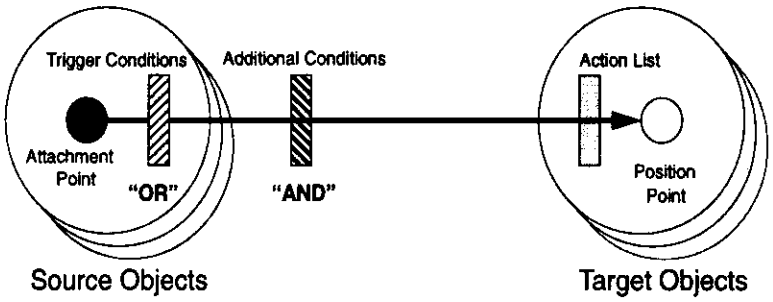

Figure 13.30: *Construction of a link.*

source can be MHEG objects, as well as virtual views. The performance of a link is always dependent on a condition (*trigger condition*), which can be expressed through the possible state transition in the source object. Only if this condition is satisfied, other conditions (*additional conditions*) are checked. If all conditions are satisfied, the link is active. In this case, the action objects, specified in the *action list* of the link, are sent to all destination objects.

Although the standard does not specify the implementation, it may be appropriate to mention that the link is checked only if a state transition of the particular source objects occurred. This implementation is driven by efficiency reasons. The *attachment point* is used to position destination objects relative to the source object. It means that coordinates in an action object express a relative position of the

destination object with respect to the source object.

Script Class

Another possibility to determine the behavior of objects or the run of a presentation is the *script class*. This class was considered to support an MHEG presentation in other run-time environments (e.g., Script/X), external programs (e.g., a C-program) or functions calls. Comparable with the content class, different languages either standardized (MHEG-Catalogue) or non-standardized (Non-MHEG-Catalogue), are supported.

13.5.5 User Interaction

Using the previously described classes, the user cannot interact with a running presentation. The introductory scenario has discussed that the MHEG standard distinguishes two kinds of user interactions: selection and modification.

Selection Class

The selection class provides the possibility to model an interaction as a selection of a value from a pre-defined value set. The explanation of the link class has shown that the run of a presentation is controlled by the occurrence of events which are given by the standard. With a selection object, it is possible to consider user interactions also in the form of such events. A selection object takes over the definition of these application-specific events. A corresponding event value is prepared for the particular selection possibilities. At a certain point in the user interaction, there exists a virtual view on the selection object (similar to a content object). The event of the user interaction is stored in the view. The storage is performed through the assignment of the particular event value onto a *selection status* field, which is pre-determined. The change of this status field should lead to the following situation: the condition of a link object is satisfied and therefore actions are sent to other objects.

The change of the status field, as well as the display and control of proper primitives (the following primitives are specified in the standard: *menu, pull-down menu, pop-up menu, button list, key list, device list, scrolling list, switch, item, button, key, device*) at the user interface are suborders of the user interface services. With the coding of a selection object, it is not specified which kind of primitives should be used for the interaction. The primitive and its properties (e.g., text on the knob) are set through actions (see selection style presentability) which are sent to the virtual views of a selection object.

Modification Class

The second form of user interaction serves as the input and manipulation of data. In contrast to a selection object, no value set is pre-defined by a modification object. The event of an interaction is represented through an entered content object. Its content is represented through the virtual view, which is defined for the content object, and it is modified through the primitive of the user interface. The actual processing state of the content object is captured in these virtual views in a *modification status* field before the modification (*modifiable*), during the modification (*modifying*) and after the modification (*modified*). Hence, it is possible, similar to the selection, to query the status field through a link object, and therefore to control the run of a presentation.

During the international agreement process, it was recognized that the ASN.1 coding is incomplete in the CD-document. Exactly, the deal concerned a variable in the content object. The value of this variable can serve again as a parameter for an action (e.g., volume by *set volume*). In the current ASN.1 coding, a reference to this variable for action objects is not possible. Therefore, so-called *generic values* are included in the later version of the standard. They can contain the event of a modification according to type. The following types have been specified: *Boolean, Character String, Numeric Value, Numeric Vector, Spatial Vector, Temporal* and *Volume*. Further, they can be reused by action objects. In some cases (e.g., fill out a sheet), it will be necessary to return the values to the application. They can be performed by the action *return*.

13.5.6 Container

Composite Class

The composite class has the task of composing all the necessary objects from the previously described classes into a presentation. Independently, if a single object is included or referenced, each composite object behaves as a *container*. It means that it represents a closed unit for the exchange of presentations among systems. The functionality of a composite object must satisfy some assumptions for the synchronized output through the MHEG engine. Through pointers (compare hypertext links) between different composite objects, any complex presentation can be created. At this point, the meaning of the abstract classes *Behavior* and *Component* (Figure 13.26) should be clear. Figure 13.31 shows graphically the structure of a composite object, including defined virtual views (*Presentables*) and two link objects (*Container-start-up, Presentation-start-up*).

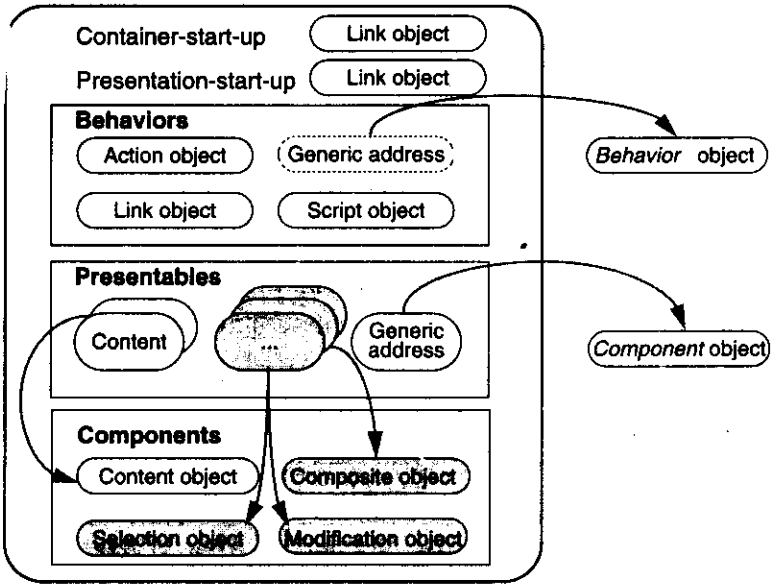

Figure 13.31: *Construction of a composite object.*

Before we describe the individual parts, let us mention that the performance of

each composite object, as by all other MHEG objects, runs in two phases. The first phase, started by the action *prepare*, initializes the composite object. It means that the object itself, both start-up-link-objects and the included virtual views, are registered. The second phase is the actual performance of the presentation. It is triggered by the action *run* on a *composite-presentable*.

This *two-phase* processing can also be seen in the construction of a composite object in both start-up-link-objects. The *container start-up-link* serves as the initialization of the *components* included in the composite object. During the initialization, eventually necessary resources for the output are activated, or reserved in the executing system. This is done to prevent, for example, delays which are caused by necessary *loading* of swapped out parts. At the same time, further link objects can be initialized which are necessary for the description of the behavior of the component object and its virtual views. Here, the link object always has the same condition per definition: "if the preparation status of the corresponding composite objects switches from *not ready* to *ready*, then ...". What happens in individual cases is in the responsibility of the application.

Also the *Presentation Start-up-link* has a pre-defined condition: "if the presentation status of a composite-presentable switches from *not running* to *running*, then ...". This condition serves as the initialization of the actual presentation, for example, through the start of a virtual view defined in the composite object.

The structure *behavior* includes all MHEG objects which prepare the component objects for the definition of the presentation's run and define the relations among the virtual views. Individually, action, link and script objects are taken in. The behavior objects can be included, as well as referenced.

The virtual views are defined in the structure *Presentable*. From a coding viewpoint, each virtual view is represented as a pair: an unambiguous integer value in the composite object and a pointer to a component object. The pointers usually point to objects in the structure *Components*. If a virtual view is related to a *generic address*, the component object is a referenced component object. This is meaningful if a hypertext structure should be constructed through the link of several composite objects

All component objects, which are listed in the structure *Components*, are contained physically in the composite object and marked with an unambiguous index.

The MHEG standard does not assume any granularity of the single composite object. A composite object can relate to, for example, single user interaction as part of an application, or it can describe a complete presentation. The granularity is determined at the end by the particular application. For example, in the case of a kiosk application, a typical page-oriented construction of the presentation's run leads to the definition of a composite object per page.

13.5.7 Closing Comments

Comparing ODA and SGML, SGML defines only a syntax for text marking, and the semantics are undefined; ODA includes a specified semantics for description of documents.

ODA offers the possibility to implement an open standard with integration of continuous media. This would allow the exchange of multimedia documents in the same way as we exchange text documents through the mailing systems today. But, there are still many missing aspects as described in the previous section.

ODA will have to consider *security* and *color* aspects. Further, backwards compatibility should be preserved. In the future, besides text and graphic, also tables and data will be supported in documents. This will require a data exchange between a document and spreadsheet, as well as a transformation from data to text. The notion of partial documents should be introduced. *Partial documents* are incomplete documents which include external pointers. These documents allow the definition of a document above the computer boundaries. Formulas should be included as part of ODA. A version management should be introduced; further content architectures for others should be defined.

MHEG provides a specification for documents which includes time and user interactions. Pictorial-related formats exist. *ScriptX* from Kaleida is the most prominent example. At this point, it is not clear which architecture/language/format will be widely accepted in the future because, for example, HyTime as an exclusion to SGML has also been developed.