

# An Object-Oriented Multimedia Database System for a News-on-Demand Application\*

M. Tamer Özsu  
Duane Szafron  
Ghada El-Medani  
Chiradeep Vittal

Laboratory for Database Systems Research  
Department of Computing Science  
University of Alberta  
Edmonton, Alberta  
Canada T6G 2H1  
{ ozsu, duane, ghada, vittal }@cs.ualberta.ca

## Abstract

We describe the design of a multimedia database management system for a distributed news-on-demand multimedia information system. News-on-Demand is an application that utilizes broadband network services to deliver news articles to subscribers in the form of multimedia documents. Different news providers insert articles into the database, which are then accessed by users remotely, over a broadband ATM network. The particulars of our design are an object-oriented approach and strict adherence to international standards, in particular SGML and HyTime. The multimedia database system has a visual query facility which is also described in this paper. The visual query interface provides three major facilities for end users: presentation, navigation and querying of multimedia news documents. The main focus, however, is querying of multimedia objects stored in the database.

**Keywords:** database management, SGML, HyTime, object-oriented design

## 1. Introduction

One of the characterizing features of multimedia information systems is their integration of large amounts of complex structured data. This characteristic makes them an excellent candidate for the use of database management system (DBMS) technology. Unfortunately, it is still rare to find multimedia information systems that use DBMSs. This precludes the system support for standard DBMS functions such as querying, update control through transactions, etc. Since most of the current generation of multimedia systems are single user systems on personal computers, this has not yet become a major problem. However, as next generation multi-user systems are developed (such as news-on-demand, collaborative and interactive work, electronic publishing) the need to develop multimedia DBMSs that provide native support for these functions are likely to increase.

---

\* This research is supported by a grant from the Canadian Institute for Telecommunications Research (CITR) under the Networks of Centres of Excellence program of the Government of Canada.

Another reason why DBMS technology has not so far penetrated this application area is the unsuitability of the relational DBMS technology for the task at hand. We defer to Section 4 the detailed discussion of the shortcomings of relational DBMSs in supporting multimedia information systems. Briefly, relational systems are good at supporting business data processing applications, but not very appropriate for supporting “advanced applications” such as multimedia information systems. Therefore their role has been restricted to the storage and management of meta-information (i.e., almost a directory service) rather than multimedia data. The emerging object-oriented DBMS technology (Dogac et al. 1994) is specifically targeted for these application domains.

We place emphasis on the use of DBMS technology in support of multimedia information systems despite the existence of a number of “multimedia *file systems*”. One reason for this is the standard argument in favor of DBMSs: file systems leave to the user the responsibility of formatting the file for multimedia objects as well as the management of a large amount of data. The development of multimedia computing systems can benefit from traditional DBMS services such as data independence (data abstraction), high-level access through query languages, application neutrality (openness), controlled multi-user access (concurrency control), fault tolerance (transactions, recovery), and access control. A second important reason is that multimedia objects have temporal and spatial relationships such as the synchronization and display of information between captioned text, video and sound. These relationships should be modeled explicitly as part of the stored data. Thus, even if the multimedia data is stored in files, their relationships need to be stored as part of the meta-information in some DBMS. As indicated above, this has been the traditional role of DBMSs in multimedia information systems; the term “multimedia database” often refers to a centralized directory service for data stored in various file systems. Finally, multimedia applications are generally distributed. Both the target application (news-on-demand) and many other multimedia applications require multiple servers to address their storage requirements. Thus, distributed DBMS technology (Özsu and Valduriez 1991) can be used to efficiently and transparently manage data distribution; distributed file systems are no match for distributed DBMSs in their functionality.

In this paper, we describe our design of an object-oriented multimedia information system design to support a News-on-Demand application. At the center of this facility is the design of a multimedia type system that allows high level modeling of multimedia applications. A second area of focus is the development of a visual querying facility. Most of the current multimedia user interfaces only support browsing. However, as multimedia databases grow larger and more complex, the need for ad hoc querying will become more prominent. Therefore we have decided to focus on these two central database management issues early on and these are the focus of this paper.

In addition to the central use of object-oriented DBMS technology as discussed above, another feature that characterizes our work is its strict adherence to the *Standard Generalized Markup Language (SGML)* and the *Hypermedia/Time-Based Structural Language HyTime* standards (ISO 1986; ISO 1992). These are ISO standards (numbers 8879 and 10744) that are sufficiently rich to support the target application, and are gaining widespread popularity. SGML mostly deals with textual documents whereas HyTime adds support for hypermedia documents (e.g., links, video, etc.).

Our work is part of a larger project on Broadband services which involves Canadian universities and research institutes. Broadband services is one of the six major projects that the Canadian Institute of Telecommunications (CITR) undertakes. CITR is one of the Networks of Centres of Excellence funded by the Government of Canada. Further information on CITR and its constituent projects can be found on the World Wide Web at

In this paper, we assume some rudimentary familiarity with object-oriented technology. We do not provide detailed descriptions of SGML and HyTime either, even though we summarize those features of these standards that are central to our design. In Section 2, we start with an overview of the target application, News-on-Demand. Section 3 discusses the system architecture that we are developing. Sections 4 and 5 are central to the paper and present the design of the type system and the design of the visual query interface, respectively. We compare our work with some of the more important design efforts in Section 6. Finally, in Section 7, we provide a summary of the current state of development and indicate the directions that we are following.

## 2. Application Environment

### 2.1 The News-on-Demand Application

News-on-Demand is an application which provides *subscribers* (or *end users*) of the service access to multimedia documents (news articles) that are inserted into a distributed database by *news providers* (or *information sources*). The news providers are commercial news gathering/compiling organizations such as wire services, television networks, and newspapers. The news items that they provide are annotated and organized into multimedia documents by the *service providers* (who may also be news providers). The subscribers access this multimedia database and retrieve news articles or portions of relevant news articles. This is typically a distributed service where clients access the articles over a broadband network from distributed servers (see Fig. 1).

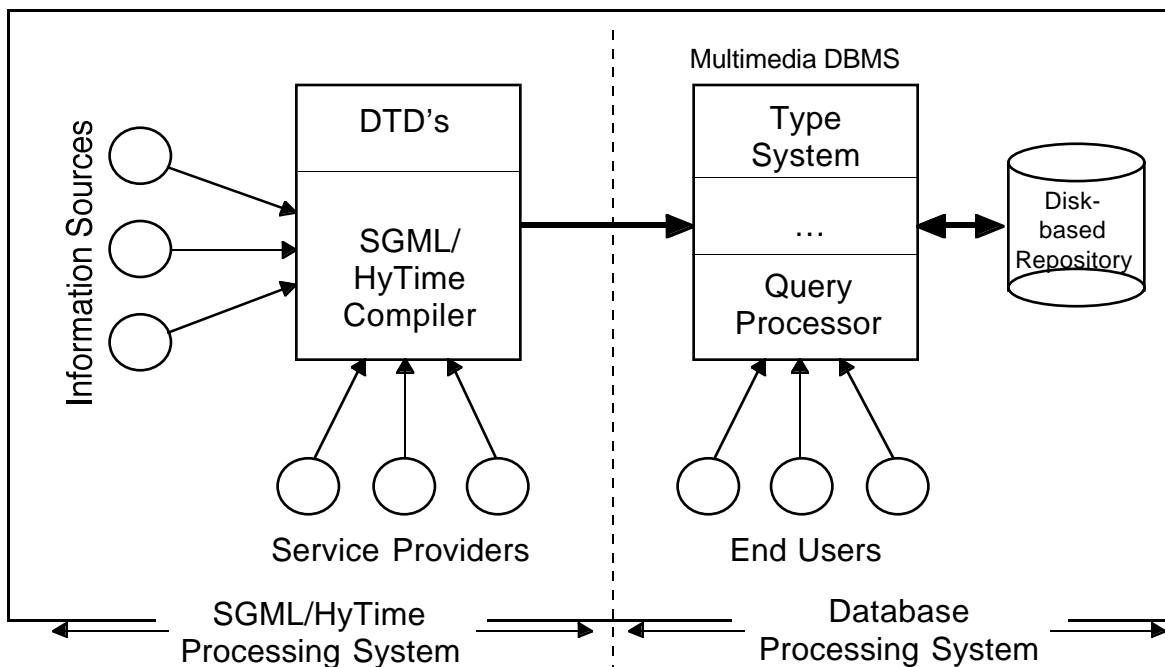


Fig. 1. Processing Environment

The News-on-Demand application raises two important issues that are not common to all multimedia systems that use databases:

- There are several news providers inserting documents into the database from different remote sites, over a network. This requires an open system following a standard for news article representation and encoding to enable transmission over the network and insertion into the database. There is a similar concern at the user's end, where different browsers and interfaces may be used to access the articles.

The choice of SGML/HyTime as the standard for document representation is reflected in the overall organization of the news-on-demand multimedia information system application (Fig. 1). News providers compose hypermedia articles on their own authoring systems. These articles are then translated to the SGML/HyTime representation. A SGML/HyTime compiler checks the document being inserted against the *document type declaration* (DTD) which describes the acceptable document structure. It then instantiates the appropriate objects in the database. Subscribers use a querying interface to access articles and/or article components from the database, which can also be queried by various system components (e.g., the quality-of-service negotiation module (Hafid et al. 1994), the synchronization module (Lamont and Georganas 1994) to obtain relevant meta-information. Our current focus is on the database processing side of Fig. 1.

- Once inserted into the database, the news article is not updated by either the news provider or the subscriber. Thus, we have a *read-only* model for the database. The news provider may insert newer *versions* of the news article, however, as time progresses. The database management system would handle the version management issues.

## 2.2 Multimedia News Documents

A document is a *structured* collection of pieces of information related to a particular subject. In a multimedia document, these pieces of information are not restricted to conventional text, but include other media such as audio, video, and images. These media may themselves be composite, so that we may have combinations of audio and video, image and text, etc. The structure of the document (i.e., the relationships between various document components) enables the contents of the document to be understood by the reader. The structure is strictly hierarchical in nature, with the document itself sitting at the root of the tree. As an example, a book is made up of chapters; chapters consist of sections; sections consist of paragraphs, and so on. This structure is in addition to the actual content of the book. In other words, there is a distinction between the document content and the structure of the document.

Two types of structure can be identified: the *logical structure* and the *presentation structure* of the document. The logical structure refers to the logical organization of document components; the presentation structure refers to the layout of the components actually displayed to the reader. The logical structure of a book would be the organization into chapters, sections, paragraphs and so on; while the presentation structure has information on the number of columns of text used to display the document, the fonts and font sizes used to display the chapter titles, whether images are displayed in color or in grayscale, etc.

Documents often have links to other documents or document components. Common examples of such links in paper based documents are bibliographic references, footnotes and cross-references. Text overlaid with a link structure is called *hypertext*. In the case of multimedia documents, this term is changed to *hypermedia*. Our model of a news article is a structured hypermedia document.


## 2.3 A Sample Multimedia News Article

This section describes a sample multimedia news document that will be used as a

running example throughout this paper. We use an article about the Department of Computing Science at the University of Alberta. The article is organized as a series of news releases which are interlinked. We will describe the document components in terms of the media present in the document; the full document is depicted in Fig. 2.


## Department of Computing Science


The Department of Computing Science at the University of Alberta is one of the oldest computer science departments in Canada, having been established in 1964. The Department is part of the Faculty of Science together with seven other departments. Its main office is located in 615 General Services Building.




*GSB - Home of the CS Department*

This is a young and active Department. It is currently made up of 32 faculty, 27 support staff and approximately 100 graduate students. There are research programs in many areas of computing science. Research ties exist with *TRLabs* and *Alberta Research Council*.

Chair's Welcome 

Tour of Facilities 

Research Programs 

M.T. Özsu

10 November 1994

Fig. 2. Sample News Document Presentation

- The **text** portion consists of the title, the (optional) subtitle, the keywords, an (optional) abstract paragraph, the date and location of the news release, the paragraphs that make up the article's content, the author, and the titles of any images appearing in the text. This information contains data that may not be shown in the presentation of the document, such as keywords.
- The **images** in the document are any pictures related to the subject of the article. In this case, the picture of the building which houses the department is included in the document. The image can be stored in any format (GIF, TIF, JPEG, etc.). The presentation of the image is also independent of the logical structure, because we may choose to reproduce the image inline with the rest of the document, or display it in a separate window.
- The **sound** or audio component of the document is the recording of a welcome message from the Chair of the Department. Here again, the representation format is independent of the logical structure of the document. The tone and volume of the audio playback are examples of presentation attributes.
- The **video** component is a tour of the facilities. The representation format of the video data (MPEG, MJPEG, Quicktime, etc.), and the presentation aspects (frame rate, size

of the window, etc.) may not be information relevant to the logical structure of the document. Video is seldom displayed on its own – there are associated media played back, or synchronized along with the video. Therefore, in the video clip about the facilities, the voice of the commentator is synchronized with the video so that the viewer does not find the lip movements out of phase with the sound of the voice being played back. There could be text subtitles displayed along with the video, giving the French translation of the commentary.

- The subscriber typically would like more information on the various events and people mentioned in the article that may not be found in the document itself. By providing **links** to other documents, or document components where further information can be found, this document enhances its information capacity. Another possibility is that the user may want to make comments (annotations) on the text that would be visible the next time the document is retrieved.

In Fig. 2, the links to other documents are marked by underlined text. There could be other more obvious icons used to denote the links. This may depend on the preferences of the viewer or author and the capabilities of the display terminal. Again, this is a presentational aspect that is separate from the logical structure of the document.

It is important to note that Fig. 2 represents only one *possible* ‘rendition’ of the news article. The user, for example, may prefer not to see any text at all, or if the available display is an ASCII terminal, only the text portion may be presented, causing the system to skip the retrieval of the image, audio, and video components of the documents.

### 3. System Architecture

The current prototype of our multimedia DBMS is an extension of a generic<sup>1</sup> object-oriented DBMS called ObjectStore (Lamb et al. 1991). The extensions provided by the multimedia DBMS include specific support for multimedia information systems. The conceptual architecture, omitting many components not yet developed, is depicted in Fig. 3. The development of a type system that supports common multimedia types is at the heart of the multimedia extensions. Our research has so far focused on this central issue as well as the development of a compatible visual query processing interface. These two components enable high-level modeling and access capability for application developers and end users. Future work, as discussed in Section 7, includes the development of an application-independent API<sup>2</sup> and a more powerful query model that supports content-based queries of images and video, as well as an optimizer for these queries.

The fact that we are currently using a generic object-oriented DBMS introduces some important restrictions. There is no native multimedia support and there is no access to source code. Therefore, the only way to extend the generic DBMS is to use standard object-oriented techniques to build a multimedia layer. Our generic object-oriented database will eventually be replaced by our own object-oriented database in later stages of this research. This will enable us to take advantage of advanced features like temporal models that are fundamental for multimedia applications. It is hoped that one of the results of this research and other similar projects will be to convince commercial object-oriented DBMS

---

<sup>1</sup> In the sense that it doesn’t have native multimedia support.

<sup>2</sup> “Application independent” in this context does not mean that the API is general enough to support *any* application. It means that the API is not tied to one multimedia application, but may be used by a number of multimedia applications.

vendors of the utility of advanced object-oriented capabilities.

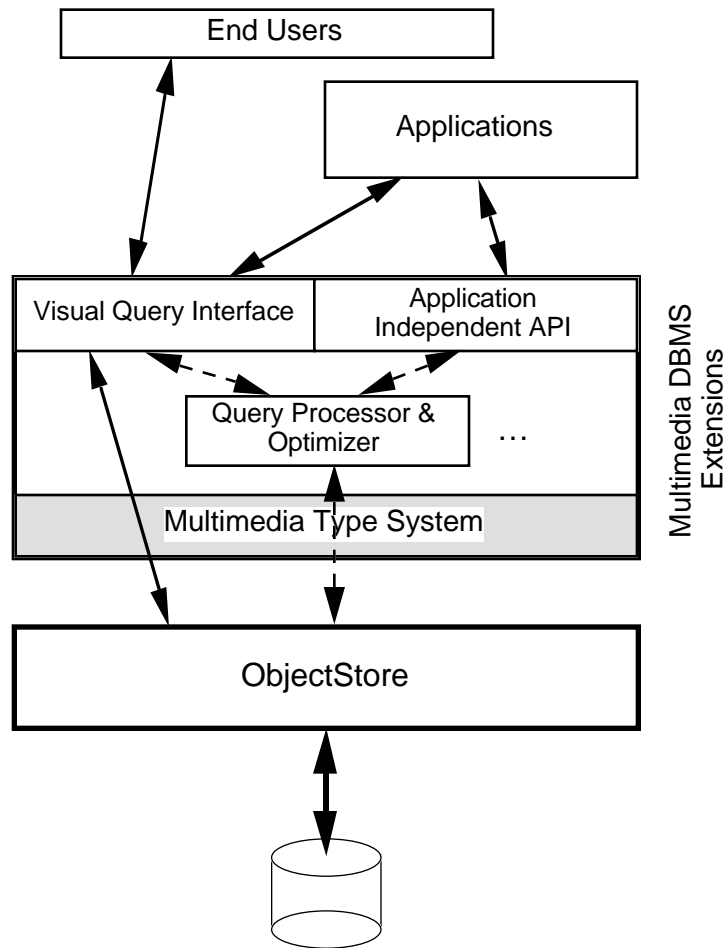


Fig. 3. Conceptual DBMS Architecture

Currently, the visual query interface – described in Section 5 – interacts directly with the ObjectStore query processor via the multimedia type system. Each menu item is linked to an ObjectStore query which is invoked when the selection is made. As our application-specific query processor and optimizer development progresses, the visual query interface will interact with it rather than with the ObjectStore system. The new interaction is shown by a dashed line.

This architecture is open so that it can accommodate various multimedia servers. Many of these servers are file system servers without full database management functionality (e.g., querying). If file system servers are used, but the applications require database functionality, then a multimedia DBMS layer can be placed on top of the file system servers and the underlying storage system can be modified accordingly.

As indicated earlier, this is a distributed system where a number of clients access a number of servers over a broadband network. In our prototyping environment, the clients and servers are IBM RS6000/360 interconnected by a broadband ATM network. This is a multiple client/ multiple server system.

Primitive media types (*monomedia*) is classified as *continuous media*, or *non-continuous media*. Continuous media refers to those types which have to be presented at a particular rate for a particular duration of time. These include audio and video. Continuous media support creates some of the most difficult problems in multimedia information systems and significantly influences the design and the load of systems. Non-continuous media such as text and still images do not have the real-time constraints of audio and video. In our system, continuous media and non-continuous media are stored on different servers. Thus, data is distributed between a number of non-continuous media servers (NCM servers) and a number of continuous media servers (CM servers). The distribution of data is transparent to the users since they use querying facilities provided by client DBMS modules, rather than directly accessing individual servers.

The current implementation does not integrate the continuous media servers with the database. The continuous media server is a disk array-based file system (Neuhold et al. 1994). In addition to the text and still images, the database stores all meta-information about the files on the continuous media file server. Finally, the database stores descriptive information about the environment that is used by the *Quality of Service (QoS) Negotiator* (Hafid et al. 1994) and the *Synchronization* routines (Lamont and Georganas 1994). The database is queried by the client modules to determine the location of a particular piece of multimedia data. After obtaining the file name and the server on which it resides, the file is accessed directly from the file server. This architecture is necessary since the database system chosen for the implementation of the application does not provide any native support for continuous media. In later versions of the system, the two components will be more tightly integrated.

The client machines contain the query interface, the multimedia DBMS client, synchronization modules, and the decoders for MPEG and Motion JPEG data streams.

The retrieval of a document involves several system components and each must access the database to determine information necessary for the completion of its tasks.

Briefly, the subscriber browses the database through the *Visual Query Interface* described in Section 5 and then chooses a document to be displayed. The subscriber then uses the *QoS Negotiator* to select the desired level of quality and cost of access. The *Synchronization* component then takes over by coordinating the delivery of several streams of monomedia data over the network. To do this, it requests the *CM Servers* and the *NCM Servers* (i.e., the ObjectStore DBMS) to retrieve the appropriate files and start the streams.

## 4. Design of the Multimedia Type System

The design of the type system actually involves the conceptual design of the multimedia database. There are four issues in designing a multimedia database:

- The different media components of the document (i.e., text, image, audio, and video) need to be modeled and stored in the database. These are called *monomedia objects* and their storage structures in the database is critical for good performance.
- A representation is needed for the document's logical structure. Not every multimedia information system represents the document structure explicitly. For example, a multimedia system that uses postscript files for text documents containing images, ignores the hierarchical structure of the document. It is important to represent this structure explicitly both for querying and for presentation.
- In multimedia documents, one has to deal with the representation of the spatial and temporal relationships between monomedia objects. These relationships are important for presentation purposes.



- The meta and descriptive information necessary for the operation of the system components needs to be determined and stored in the database. As well, access routines need to be provided (as part of the API) for easy access to this information.

In this section, we focus on the first three issues which are central to the database design. The following three sections present our approach to addressing these issues. The meta and descriptive information that is stored in the database is described in (Vittal et al. 1994). As indicated earlier, we use an object-oriented approach and follow the SGML/HyTime standard. A few words about our design decisions are in order.

We use object technology – rather than relational – for a variety of reasons. First, multimedia objects are complex in their structure. The *primitive objects* (monomedia objects) are not only simple strings or numbers (e.g., names, addresses, and salaries of employees), but also include video, digitized voice and images. There is no support for these types in relational systems nor is there a way to extend the type system to incorporate them (extended relational systems are an exception). The “binary large objects” (BLOBs) that are supported in some relational systems are not sufficient to model these entities. One can store the image, for example, as one BLOB, but it is not possible for the relational DBMS to interpret this BLOB (i.e., access parts of it or perform image-specific operations on it). Object-oriented DBMSs, even though they may not support these types generically, can at least be extended to include them as part of the multimedia DBMS extensions.

Second, multimedia documents are structured complex objects containing a number of these primitive objects. For a database where such multimedia documents are stored, there should be facilities for (a) accessing objects based on their semantic contents, and (b) accessing different components of these objects. Furthermore, there are relationships among the multimedia objects (i.e., classification, specialization/generalization, and aggregation hierarchies) that need to be modeled (Dimitrova and Golshani 1992).

Third, multimedia information systems require an extensible data model that allows application designers to define new types as part of the schema. Furthermore, the applications themselves must be able to add and delete new multimedia types dynamically. Therefore, multimedia systems must not have static schemas and the DBMS must be able to handle dynamic schema changes. Object-oriented systems meet all of these requirements much better than relational ones.

We follow an international standard for multimedia document representation, because the target application demands that a standard representation be used, for which various authoring tools are available. The tools themselves can be different, but they should at least be based on the same document representation. This is one way to support heterogeneity of tools while providing a unified database representation.

SGML (ISO 1986) has been chosen as the standard to follow because of its suitability for the target application, its relative power, its widespread use (for example, the Hypertext Markup Language, HTML, that is the basis of World Wide Web is an application of SGML) and its role as the basis of the HyTime (ISO 1992) hypermedia representation standard. SGML mostly deals with textual documents whereas HyTime adds support for hypermedia documents (e.g., links, video, etc.). The two other alternatives to follow would have been *Office Document Architecture (ODA) Standard* (ISO 1989) and the MHEG Standard. ODA is not sufficiently rich to be used in this application and the MHEG standard (even in draft form) was not yet released when this work was started. While SGML/HyTime is gaining acceptance and tools are being developed for it, MHEG is still in draft form.

## 4.1 Modeling of Monomedia Objects

Since the continuous media file server is not yet integrated with the multimedia database, we only store descriptive information about audio and video objects in the database. Text and images are stored in the database. Since ObjectStore does not provide native support for multimedia data, the multimedia DBMS that sits on top of ObjectStore implements these data types as *atomic types*.

### The Type System for Atomic Types

Fig. 4 illustrates the type hierarchy for atomic types. In this paper, we omit full descriptions (i.e., attributes and methods) of these types due to space considerations. They are given in (Vittal et al. 1994). Instances of atomic types hold the raw (mono) media representation along with other information relevant to the QoS scheduler and synchronization module.

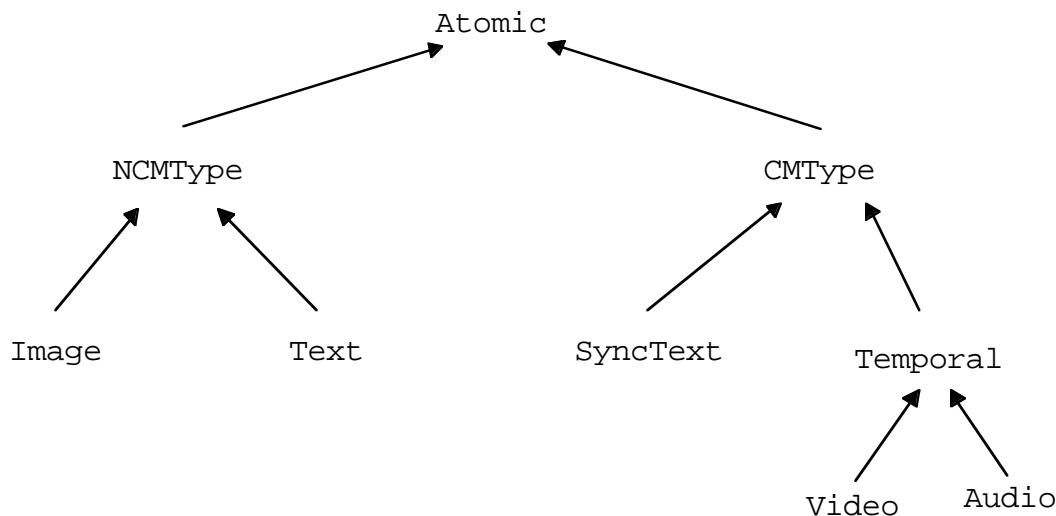


Fig. 4. Atomic Types Hierarchy

There are two subtypes of atomic media types – one for non-continuous media (NCMType) and another for continuous media (CMType). The attributes and methods which are common to both kinds of media are abstracted in the Atomic type. These are the length and generic QoS parameters such as jitter, cost and delay.

The NCMType media are further subtyped into Text and Image media types. NCMType has the attribute content which is an array of characters. The Text subtype has additional methods: match which implements a pattern matching algorithm, and substring which returns a portion of the text object given the two integers representing the start and end locations. The Image type has additional attributes such as the width, height and colors of the image. Both these types have attributes for the QoS parameters specific to the media they model. The Image type can be further subtyped to reflect the different storage formats possible.

A similar subtyping scheme is seen on the CMType side of the type hierarchy. The Video type can be subtyped to handle different storage formats. Synchronized text

(SyncText) is not subtyped from Text, since it is stored on the file system, not as an object in the database. The methods `match`, and `substring` cannot be applied to the synchronized text media. The Temporal supertype of video and audio is defined because both have a `duration` attribute. Note that the actual data corresponding to objects of type CMTYPE (and its subtypes) are stored in the continuous media file server which is not under the control of the multimedia DBMS. Thus, these objects in the database only store meta-information.

### Storage Model for Text

Text (a character string) is an atomic type which is supported in the database system. However, in the news documents, the text component of the article is richly structured, consisting of many hierarchically arranged components (also called *elements*). One alternative for representing text components of a multimedia document is to define object types for each of these structure components and associate with each of them a fragment of the complete text of the article.

Storing the text content of the article by fragmenting it in this manner can have serious performance implications. For example, to store the second instance of the paragraph element in the sample document of Fig. 2, we need three fragments – the emphasis element, the link element and the rest of the text. Accessing the text of the paragraph now involves three accesses to persistent store.

Although there are strategies such as clustering to improve performance, with large objects involved, these techniques may be inadequate. In any case, the pointer swizzling overhead of these objects cannot be overcome by clustering. Furthermore, if pattern-matching methods are defined on text elements, it would be necessary to re-assemble the entire text component of the document which has performance implications.

In addition to performance issues, there are modeling complications as well. One problem is to decide what the granularity of the fragmentation should be – paragraphs? sentences? words? The granularities can be determined by the granularities of the logical elements of the document. Thus, each logical element would contain a fragment of the text. For example, there would be an Emphasis type for instances of logical emphasis elements. This can cause several copies of the same piece of text residing in various logical element instances. The second problem which arises is as follows: suppose an emphasis starts at some position in one word and runs until some position of a subsequent word (i.e., does not cover entire words). Since there is a logical emphasis element in the markup of this document, it would be necessary to create an instance of the Emphasis type and store the emphasized text as the value of one instance of this type. However, this precludes the possibility of querying for either one of those two words involved in the emphasized string.

To avoid fragmenting the textual elements in this manner, we store the entire text content as a single string. To associate a particular instance of an element with its text content we store the first and last character locations of that portion of text in the entire text content. We call pairs of integers such as these, *annotations*. Using this model the text content of the sample news document can be modeled as depicted in Fig. 5. In this example, the first paragraph instance has the annotation [33, 338]. The link sub-element of the paragraph has the annotation [264, 274].

Every document instance in the database has a “base” object (of type Arti-

cle\_root) associated with it which stores the text string forming the text content of the article, and the lists of annotations associated with each text element type. To display the document, the browser can scan these lists efficiently and determine the presentation of the text. We map this representation to a type system by defining a type, `Text`, whose instances store a single string that is the entire text content of a document as represented in Fig. 5. We also define a type to correspond to every allowable annotation, as specified in the document DTD.

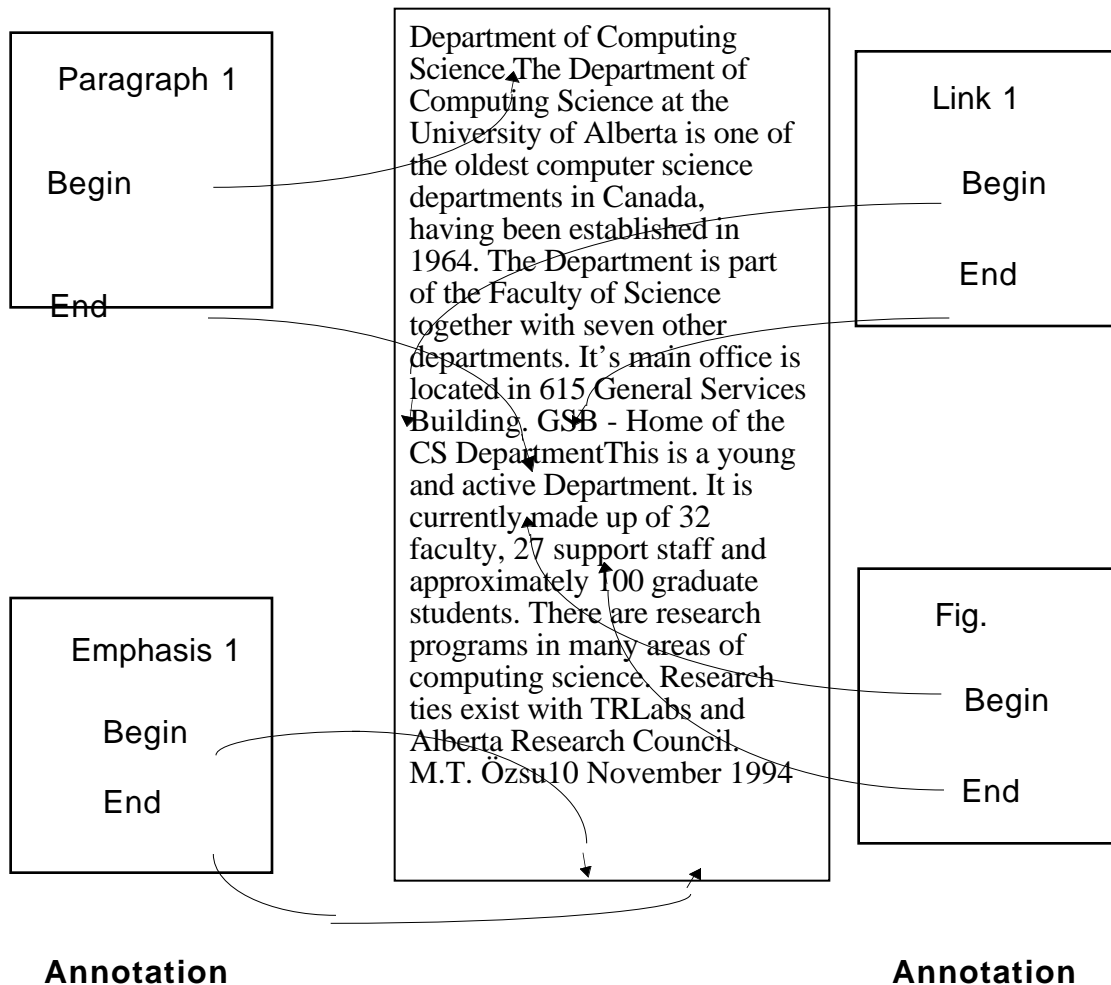


Fig. 5. Annotations to Mark-Up Text Documents

There are two distinct advantages of using this storage model for text elements:

- Displaying the text becomes faster, and more efficient because multiple accesses to persistent store are avoided.
- Indexes can be built on these annotation objects which can aid searches for element instances. For example, it is possible to search for emphasized strings in a document.

There is one disadvantage of this approach. Updates to the text content are expensive, since a change to the content of a document may cause many annotations to change. This

can be avoided to a certain extent by specifying annotations relative to some enclosing structure, say with respect to a paragraph. Then, after an edit, the only annotations that change are the annotations of the sub-elements in the edited paragraph and the annotations of all following paragraphs but not the annotations for the sub-elements of these paragraphs.

## 4.2 Modeling Document Structure

The logical structure of a document is necessary for its contents to be understood. For example, document presentation, certain queries and hyperlinks all rely on the logical structure of the document. SGML uses markups to represent this information.

### Markups, Elements, Document Type Definitions and Architectural Forms

SGML is a meta-language which describes the logical structure of a document by using *markups* to mark the boundaries of its logical elements. The *generalized markup* approach of SGML separates the description of structure from the processing of the structure. The philosophy is that processing instructions can be bound to the logical element at the time of formatting, or display. Descriptive (or generalized) markup identifies logical *elements* using *start tags* and *end tags* to mark their boundaries.

The markup in SGML is *rigorous* (Goldfarb 1990) in that elements can contain other elements to form a hierarchy. Thus, `chapter` elements can contain `title` and `section` elements; `section` elements can contain `paragraph` elements and so on. The hierarchy is a tree, and whole subtrees can be manipulated as one unit. In other words, an SGML document consists of instances of document elements arranged in a hierarchical structure.

SGML does not specify what these elements should be, or what the hierarchy should look like. Instead, the list of elements types, and the relationships between them is expressed as a formal specification called a *Document Type Declaration* (DTD). A DTD is written in SGML by the document designer for each category of document being designed. In our case, we need to write a DTD for multimedia news articles, but there could be DTDs for books, letters, technical manuals etc.

A DTD specifies *element types*, the hierarchical relationships between element types, and *attributes* associated with them. Attributes contain information that is not part of the document content. In the example multimedia news document of Fig. 2, the following element types can be identified: `article`, `headline`, `date`, `paragraph`, `Fig.`, `Fig. caption`, `emphasis`, `author`, `link`. Note that the article itself is considered an element and there may be other elements (e.g., keywords) that are not demonstrated in the rendition of Fig. 2. If we omit the audio and video elements, the marked-up sample news document looks like the following:

```
<article>
<front>
<author> M.T. Özsu </author>
```

```

<keywords> computer science, University of Alberta, education </keywords>
<headline> Department of Computing Science </headline>
<date> 10 November 1994 </date>
<location> Edmonton, Alberta, Canada </location>
</front>
<body>
<paragraph> The Department of Computing Science at the University of Alberta is
one of the oldest computer science departments in Canada, having been established in
1964. The Department is part of the Faculty of Science together with seven other
<link linkend=sci_depts.sgml>departments</link>. It's main office is
located in 615 General Services Building.</paragraph>
<figure filename=gsb.gif>
<figcaption>GSB – Home of the CS Department</figcaption></figure>
<paragraph> This is a young and active Department. It is currently made up of 32
<link linkend=faculty.sgml>faculty</link>, 27 <link
linkend=faculty.sgml>support staff </link> and approximately 100 gradu-
ate students. There are research programs in many areas of computing science. Re-
search ties exist with <emphasis>TRLabs </emphasis> and
<emphasis>Alberta Research Council</emphasis>.</paragraph>
</body>
</article>

```

This document is declared to be an article type. Thus, the legality of its mark-up is determined according to the article DTD which defines the acceptable article document structure. The full DTD for the multimedia news articles is given in Appendix 1.

The discussion so far omitted links, audio, and video objects. These are the domain of the HyTime standard which defines 69 special hypermedia elements, called *architectural forms* (AF), that can be used in DTDs. For example, there is an architectural form called

click which defines a so-called *contextual link*. A contextual link is a link with an anchor rooted in a particular context, exactly like the links shown in the sample news document. To use architectural forms in our HyTime document instances, we first define element types which **conform** to the specification of the architectural forms. Then we use instances of these conforming element types.

### Type System for Elements

Fig. 6, 7 and 8 show the type hierarchies for logical document elements. The supertype of all elements is the `Element` type. This models the fact that *all* elements need to maintain a reference to their parent element in the document instance hierarchy, so that the hierarchy can be navigated starting from any element. When links are made to arbitrary elements in different documents, or when searches are performed over several documents, it is often useful to know the article these element instances belong to. Therefore, each element maintains a reference to the article that contains it. `Element` is subtyped into `TextElement`, `Structured` and `HyElement`.

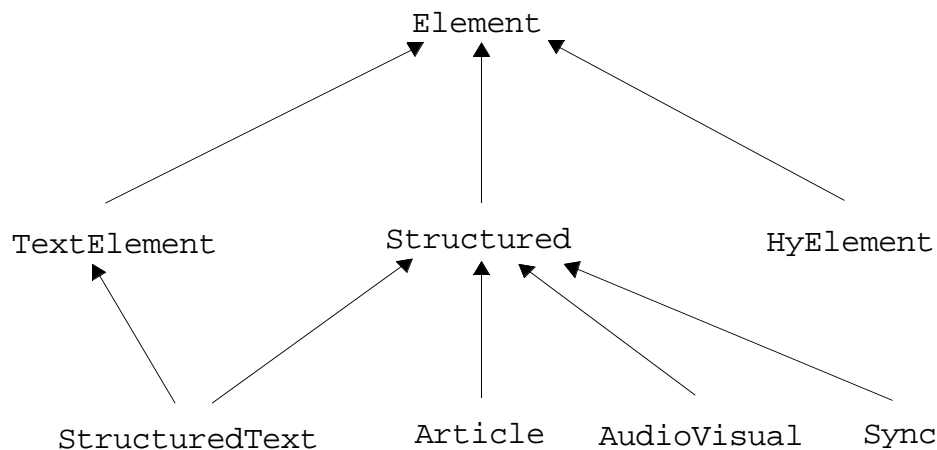


Fig. 6. First-Level Element Type Hierarchy

In the DTD for news documents given in Appendix 1, we divide the document into components called `async` and `sync`. This reflects the fact that continuous media with synchronization constraints (`sync`) need to be handled by HyTime conforming element types, and other SGML element types are adequate to deal with text and image data (`async`). The supertype `HyElement` encompasses all the HyTime elements used in the DTD. We delay a discussion of these types until Section 4.3.

Due to the annotation-based storage model, elements defined for textual data in the DTD have corresponding types in the type system each with an attribute whose value is the annotation of the element in the article instance. Their supertype is the `TextElement` type. This supertype has methods to manipulate these annotation values. An additional method defined on `TextElement` is `getString` which returns the string value captured by the annotation. The `TextElement` type hierarchy (excluding `StructuredText` which is described later) is illustrated in Fig. 7.

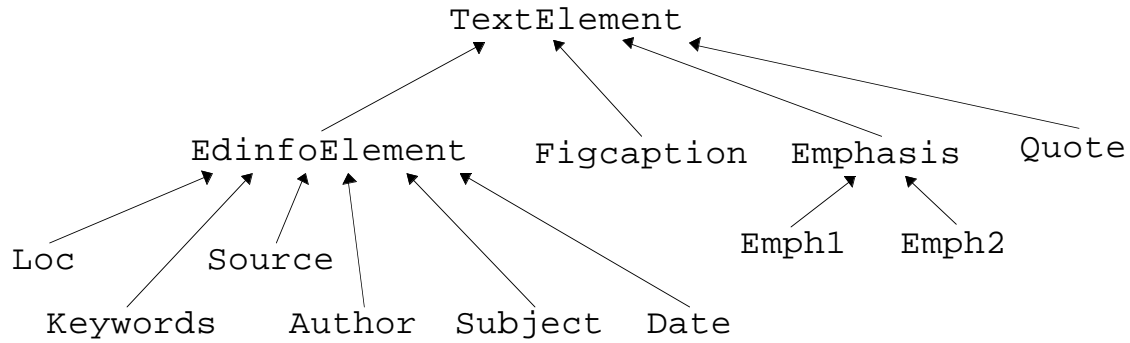


Fig. 7. Type Hierarchy for Other Text Elements

The types in Fig. 7 correspond to text elements that do not have any subelements. Most of the types here do not have any additional methods or attributes other than those present in `TextElement`; they have been created as subtypes purely for classification purposes, and to retain the uniform approach of modeling all element types as types in the type system.

The type `Structured` is a supertype for elements in the DTD with complex content models. That is, structured elements can have child elements in the document instance and need to maintain references to them. Correspondingly these elements have the method `getNth` which returns a reference to their  $n^{\text{th}}$  child element. Since the types of these child elements are so diverse, the only common supertype is `Element`, which is the return type of this method.

Elements which are both structured and based on text have a common supertype called `StructuredText`. The subtypes of this type includes all text elements with complex content models, like `list`, `section`, `Fig.`, `frontmatter`, etc. The type system rooted at this type is shown in Fig. 8.

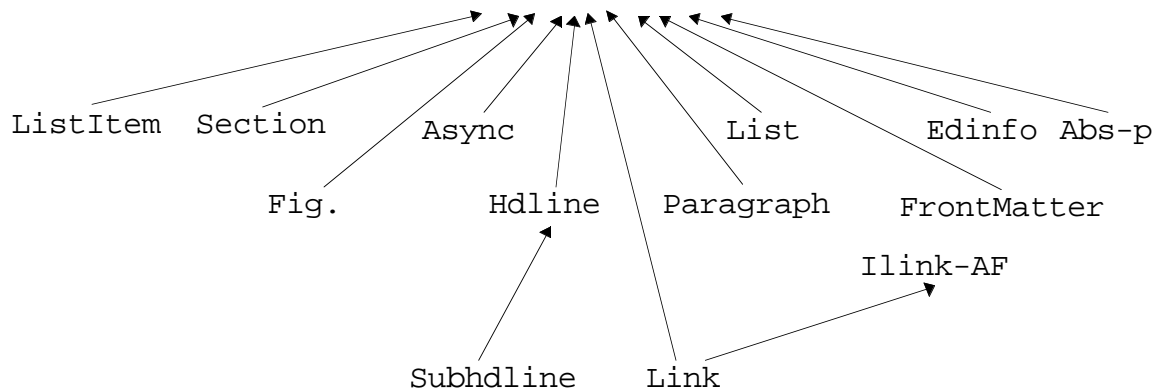


Fig. 8. Type System for Structured Text Elements

Instances of the `Article` type are at the root of the composition hierarchy. According to the DTD, they should have references to instances of `Frontmatter`, `Async` and `Sync` types. In addition, the *date*, *source*, *subject*, and *author* are attributes (type



String) of Article, even though these values are already stored (by means of annotations) as instances of Date, Source, Subject, and Author types respectively (Fig. 7). There is a performance reason for replicating the contents of these attributes. We would like to index the collection of Article instances on the values of these attributes, since queries predicated on these are likely to be frequent. However, ObjectStore collections can only be indexed on attributes. The string value of instances of Date, Source, Subject, and Author can only be obtained by the application of the method getString(). Hence, although we could have methods getDate, getSource, getSubject, and getAuthor for the Article type, it would not have been possible to build indices on these methods if we had not defined them as direct attributes of Article.

AudioVisual and Sync are the other two subtypes of Structured. In the DTD, the element audio-visual models one set of logically related HyTime components. For instance, if the document was one hour of a television broadcast, there would be one audio-visual each for the news, for the commercial segments, etc. The whole broadcast would be modeled by the sync element, and captured by the Sync type. Sync instances are collections of AudioVisual instances.

Structured elements have complex content models and pose two problems. The first problem is due to the ‘or’ connector (‘|’) in the content model. For example, the Async element has the content model:

```
<!ELEMENT async -- (section|Fig.|link)*>
```

If we have three fields for the Async type each of which is list of references of the type of one of the three elements listed on the right hand side, then we lose the relative orderings between say, Section instances and Fig. instances which are the children of the Async instance. One solution to this problem is to have just one list of references of the existing common supertype of Section, Fig., and Link; this is StructureElement in this case. However, this leads to type checking problems since references to any subtypes of StructureElement (say Paragraph elements) could now be inserted into the list.

A second solution is to use *union types*: the parameter of the list of children is the union type of the three types: Section, Fig., and Link. Unions are present in the C++; ObjectStore allows named union types to be made persistent. However, a *discriminant* method has to be provided to differentiate between the types in the union, and the user has to ensure that the right type is being accessed (i.e., the user has to do some type checking). The third solution (the one we have adopted) is to create an abstract supertype of Section, Fig., and Link. The parameter of the list is then this supertype and there are no type checking problems. The drawback is that it creates an explosion of types in the system. We call abstract supertypes created for this purpose *pseudo-union* types.

The second problem occurs in the use of the ‘follows’ connector (‘,’). For example the element frontmatter has the content model:

```
<!ELEMENT Frontmatter --(Edinfo,Hdline,Subhdline,Abs-p)>
```

This means that instances of `Edinfo`, `Hdline`, `Subhdline`, and `Abs-p` must follow each other in any document instance. To capture this in our type system, we need a mechanism to order the attributes of the type `Frontmatter`. Again, this feature is not present in the data model of `ObjectStore`. We assume an implicit ordering of attributes in this case. The behavior of the `Frontmatter` type is such that it enforces the ordering. Thus, when the method `getNth(3)` is applied to an instance of `Frontmatter`, the result is a reference to an instance of the type `Subhdline`.

### 4.3 Modeling Presentation Information

The third major issue in multimedia database type design is the representation of spatio-temporal relationships among document elements. This information is accessed by the synchronization routines in planning the retrieval of monomedia objects and the presentation of these objects according to a presentation specification. Our representation of spatio-temporal relationships is compliant with the HyTime standard.

The HyTime standard is divided into modules, each of which describes a group of concepts and architectural forms. These modules are the base module, the measurement module, the location address module, the hyperlinks module, the scheduling module, and the rendition module. Each module may use certain features of other modules lower down in the hierarchy; thus the location address module does define AF's which are used in the rendition module. Each HyTime DTD must declare the names of the modules it requires.

In our DTD for multimedia news articles, we use certain features of the base module (as do all HyTime documents), some of the location address module, some of the hyperlinks module, and some of the scheduling module. We skip the description of these modules, except for the scheduling module. Concepts needed from other modules will be defined where required.

#### Finite Coordinate Spaces

To represent relatively simple spatial and temporal constraints between document elements, we use the *finite coordinate space* (FCS) architectural form defined in the scheduling module. This, in turn, requires features of the measurement and location modules. In the discussion that follows, several architectural forms will be used in the examples but not explained. It is hoped that the relevant ideas can be abstracted. The following convention is used: whenever an element type name appears with a 'my\_' prefix in an example, then it conforms to the architectural name that follows the 'my\_' prefix.

HyTime models space and time using axes of finite dimensions. A finite coordinate space is a set of such axes. All measurements are associated with *axes*. The units of measurement along axes are called *quanta*. There are various types of quanta defined in HyTime, besides the normal units of measurement – including characters, words, nodes in trees, etc. Fig. 9 describes the various concepts used. The finite coordinate space shown here has three axes: two spatial, and one temporal.

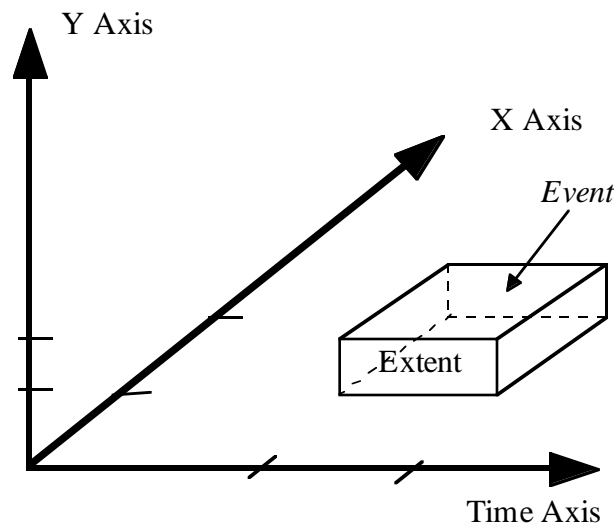


Fig. 9. Axes, Events, and Extents (adapted from (DeRose and Durand 1994))

In HyTime, an extent is a set of ranges along the various axes defining the FCS. An *event* corresponds to an *extent* in the FCS. An *event schedule* consists of one or more events. Extents are specified using the `extlist` architectural form. Events are created using the `event` AF; event schedules using the `evsched` architectural form. The document instance associates a data object with the event. The semantics and the manner in which the events are rendered are defined by the application. An example depicting the use of FCS's for storing presentation information is given in (Vittal et al. 1994). We omit it in this paper due to space considerations.

### Separation of Presentation from Logical Structure

The SGML/HyTime philosophy is to bind the processing instructions to the logical elements of the document as late as possible. Thus, the association of an “emphasis” element to italics, for example, is not done until formatting (presentation) time, even though this information needs to be stored in the database. The list of associations between logical elements and their processing instructions is known as a *style sheet*. We, therefore, store style sheets in the database.

To represent the style sheet in the database as a separate piece of information from the document instance we extend our type system to include a DTD for style sheets. The style sheet DTD that we implement is given in Appendix 2.

It should be noted that style sheets are inadequate to specify the entire range of processing instructions. One example is context sensitive processing – the processing of an emphasis element may depend on whether it occurs in the abstract paragraph or in the main body of text. Another aspect is the layout of text – for example in two or three column formats. The first can be handled using the `LINK` option of SGML (Goldfarb 1990). For the second problem, we can associate this information as processing instructions for the root of the document instance tree; in this case the instance of an article element.

## Type System for Presentation Information

Since we use HyTime to model temporal and spatial information, the same concept of a document can be extended to include the presentation layout as well. To represent processing instructions, we have another category of documents – the DTD for style sheets. This too is a collection of elements with hierarchical relationships.

The type HyElement in Fig. 6 is the supertype for all HyTime elements in the type system. Its immediate subtypes are those modeling the architectural forms used in the DTD. The attributes of HyElement are its ID (assigned by the author of the document, or by the document authoring software), and the string representing the name of the architectural form. This models the assumption that every HyTime element can be linked to, and should answer the architectural form it conforms to. The sub-hierarchy rooted at HyElement is depicted in Fig. 10.

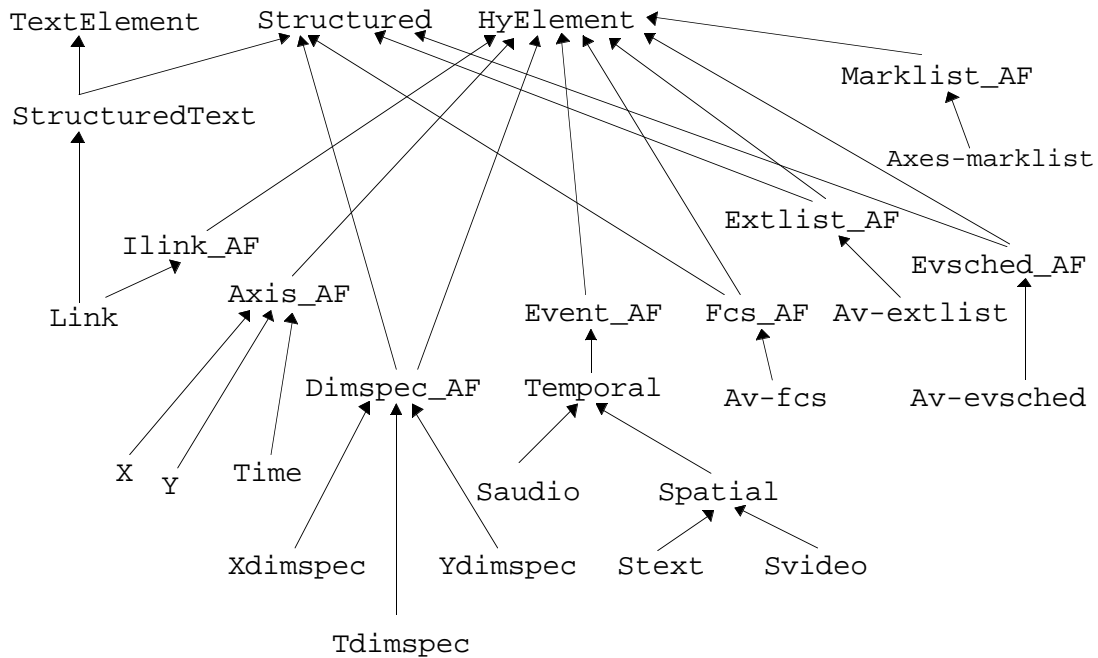


Fig. 10. Type Hierarchy for HyTime Elements

Of the nine HyTime architectural forms used in the DTD, the most important are the **fcs** and the **ilink** AFs. The **ilink** AF can be a Structured element depending upon the DTD designer. We create a type for this AF, called `Ilink_AF`, as a subtype of the HyElement type. In the DTD for news articles, the `link` element has a complex content model and conforms to the **ilink** AF. Therefore, the `Link` type is a subtype of both `Ilink_AF` and `Structured`. According to the HyTime standard, the **ilink** AF has to have the attributes `linkends` and `anchrole` (anchor role). The **ilink** AF can be used to specify multiple destinations per link, and can link any element to any other element. The `linkends` attribute is therefore a list of Element references. The `anchrole` is of type String. The `Ilink` type has the pure virtual method `traverse` which takes the object ID of a destination element (present in the `linkends` attribute), and performs a traversal according to the applications semantics (hence `Ilink` is an abstract type, like most other types representing architectural forms). This method is defined in the `Link` subtype.

The **fcs** element is important because it provides the interface to the other system components to determine the types of media objects present in the continuous media, and to determine the presentation schedule of the media objects which are a part of the **fcs**. The attributes and methods of the *Av-fcs* type illustrate how this information can be obtained. It has a method `GetSchedule` which returns an object of type `TimeFlowGraph` which contains the schedule of the objects. The method `GetVideoObjects` returns a list of references to objects of type `Video` (an atomic type). These atomic objects can be queried for location and QoS information.

The other HyTime elements (Fig. 10) are architectural forms used in the DTD. All three axes (*x*, *y*, and *time*) declared in the DTD are similar, except for the dimensions, measurement units, and measurement granularity, which are reflected in the values of the *axisdim*, *axismas*, and *axismdu* attributes. However they have different semantics in the DTD; thus they are separate subtypes of *Axis\_AF*.

*Event\_AF* type has been subtyped to represent the three different types of events possible in the finite coordinate space – text, video and audio (*S<sub>text</sub>*, *S<sub>video</sub>*, and *S<sub>audio</sub>*). The intermediate supertypes *Spatial* and *Temporal* reflect the fact that *S<sub>audio</sub>* has a purely temporal dimension, while *S<sub>video</sub>* and *S<sub>text</sub>* have both spatial and temporal dimensions. These types have attributes which reference the atomic type instances which store the media associated with these objects. For instance, an *S<sub>text</sub>* type instance will have a reference to an instance of *SyncText*. The *Exspec* attribute reference the *Extlist* instances which hold the values of the extents of these elements along the three axes.

The **extlist** architectural form has the concrete element type *Av-extlist*. The children of this element are the three elements conforming to the **dimspec** architectural form. Therefore the *Av-extlist* type is a subtype of the *Structured* type. The three subtypes of *Dimspec\_AF* (not shown in the diagram) are exactly the same, but are separate for classification purposes. They contain elements conforming to the **marklist** AF, and are hence *Structured* elements.

The elements of the style sheet DTD are shown in Fig. 11. There are 7 elements in that DTD, of which only 3 are structured elements (*style-sheet*, *rule*, and *spec*). All the elements consist of strings. It is preferable not to use the annotation model to store these text elements. This is because the size of the style sheet is small (no large objects, and a few lines of text). Therefore the types modeling the elements of this DTD are either subtypes of *Structured*, or are direct subtypes of *Element*.

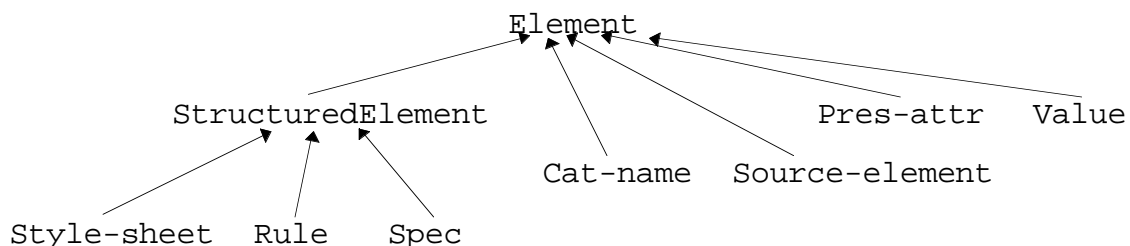


Fig. 11. Style Sheet Element Types

## 4.4 Example Design

In this section we use our sample document to demonstrate how the type system described in the previous sections can be exercised. This discussion concentrates on the composition hierarchy that emerges among objects according to the document structure. The composition hierarchy is based on the attributes of each type. Instead of presenting the attributes abstractly, we will demonstrate how the structure of the sample document is mapped to a composition hierarchy as objects are instantiated and their attribute values set. This discussion refers to Fig. 12 and 13, where object instances of type X are denoted as MyX and the arrows are from objects to their component objects.

The root of the composition hierarchy (Fig. 12) is one instance of the `Article` type object, called `MyArticle`. `MyArticle` has three attributes, among others, that point to a `Frontmatter` type object, called `MyFrontmatter`, an `Async` type object, called `MyAsync`, and a `Sync` type object, called `MySync`. `MyFrontmatter`, holds the information in the document that is delimited by the markup `<front>` and `</front>`. As discussed in Section 4.2, the body of the document is separated into an asynchronous part (`MyAsync`) and a synchronous part (`MySync`). The asynchronous part describes the text and image part of the document.

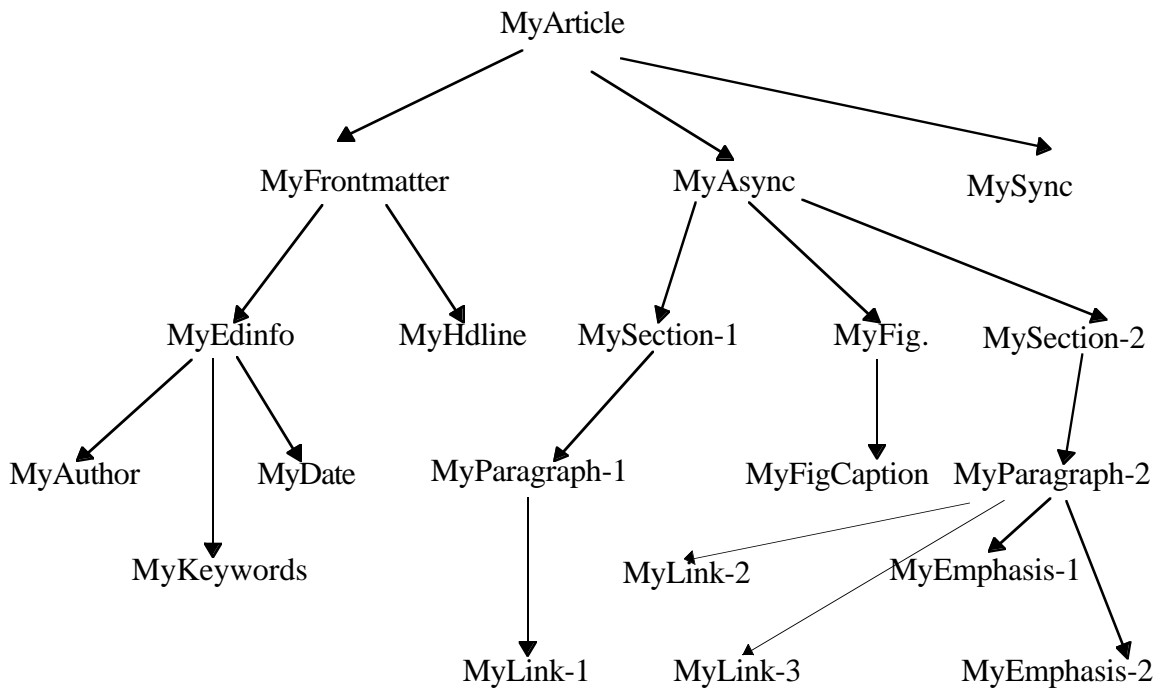


Fig. 12. Partial Object Composition Hierarchy

According to the DTD of Appendix 1, each document is separated into sections first. In our example, we assume that the `Fig.` which consist of the building's picture and the text before it is one section (even though it is only one paragraph) and the part after the `Fig.` is a second section. Thus, there are two `Section` type objects (`MySection-1` and `MySection-2`) as well as one `Fig.` type object, `MyFig.` which are components of `MyAsync`.

The rest of the hierarchy should be obvious. Note that there are composition paths

from some of these objects to instances of atomic types (Fig. 4). For example, MyFig. has a link to an object of type Image (or one of its subtypes depending on the type of the Image) for the picture of the building.

The synchronous part of the document that corresponds to the audio and video is shown in Fig. 13. In the sample news document of Fig. 2, it is assumed that a closed captioned video of the Guided Tour is associated with the article.

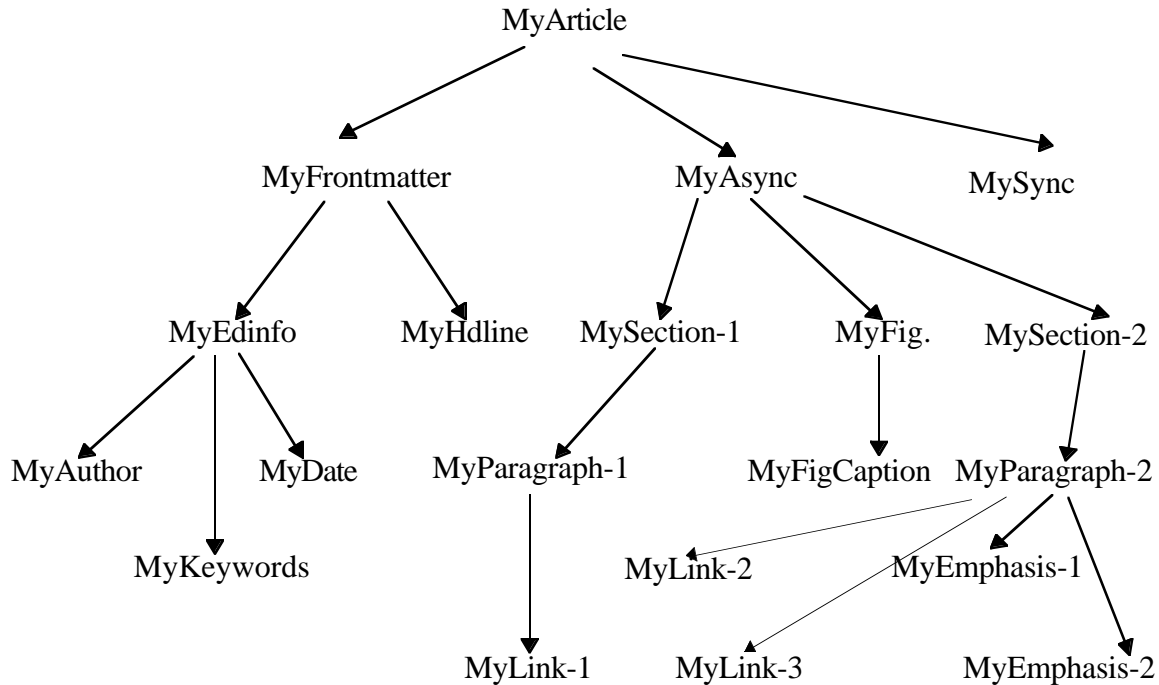


Fig. 13. Composition Hierarchy for the Synchronous Portion of the Example Document

The closed caption video consists of the video, synchronous with the commentary (audio), along with captions which appear periodically, giving the French translation of the commentary. The three media are modeled as events in the finite coordinate space described in the DTD. The whole “audio visual” therefore consists of the two spatial axes (the time axes), the finite coordinate space, and the list of event extents along the axes.

Since there is only one closed captioned video, there is only one instance of the AudioVisual element in Fig. 13, which has as its children the instances of the axes, the instance of the Av-fcs, and multiple instances of extent lists (MyAv-extlist).

The Av-fcs instance itself contains just one event schedule (there could be several; for example if the commentary had been partitioned into logical segments). The event schedule is just the collection of the events occurring in the FCS. Since the audio and video data are not segmented, there is just one audio event, one video event; there are however several synchronized text (Stext) event instances, one for each caption.

According to the DTD, each extent list consists of dimension specifications (dimespec), which in turn consist of marker lists (list of positions along the axes). The first two instances of the Av-extlist type are shown in the figure; the contained dimespec

instances are shown for the second. We omit the marker list since it is too involved to display in one figure.

Not shown in the composition hierarchy are the occurrences of instances of atomic types. In Fig. 12, MyFigure has a reference to an instance of Image. In Fig. 13, MyAudio has a reference to an instance of Audio, MyVideo to an instance of Video, and MyText-1, etc. have references to instances of SyncText.

## 5. Visual Querying Facility

Many of the tools that access multimedia information systems are based on browsing. In the case of hypermedia documents, these browsing tools may become sophisticated enough to allow navigation via links, playing of audio and video components, etc. Many tools ignore the equally important query facility which allows ad hoc querying of the multimedia news database. Our research focuses on *querying* multimedia databases and the prototype that we have developed provides an integrated system for querying and browsing of multimedia news documents.

We are ultimately interested in the development of query languages, access primitives, and visual query facilities that would allow sophisticated querying of these databases, including content-based querying of all types of media. However, our current research and system has so far concentrated on elaborate searching of textual parts of documents and provides means for accessing other monomedia objects by means of keywords.

The following retrieval scenario elaborates on the type of queries the user and the system may perform.

- The user wishes to see some articles about educational institutions. Alternatively, the request may be to view some articles featuring University of Alberta. Therefore, the database is queried for all documents with the *keywords* **education** in them (or **University of Alberta**).
- The database returns a list of *titles* of articles with the required keywords. Other information displayed could be the list of media types in the article, and the nominal cost of retrieval of the document. This cost changes as the user negotiates the quality of service desired (or can be paid for) with the system. Note that each of these additional pieces of information is obtained through the user interface by querying the documents in the list.
- The user selects one particular article (for example, the one described in Section 2), and retrieves the document after negotiating the cost of access.
- The retrieval process itself triggers additional queries to the document in the database to fetch the necessary information for accessing and displaying the document. This includes fetching meta-information, presentation information, etc.
- Although a keyword based search is the most likely scenario, there are other queries possible that would return a list of documents matching the search criteria. For example:
  - return documents with a particular text string within the text of the article.
  - return documents with video, but no text.
  - return documents with a certain *location* and *date*.
  - return documents by a certain *author*, etc.

Searches also allow the user to retrieve components of a document (only video, for example) rather than the complete document.



- Queries can be performed on the displayed document as well. Text string matching is a common example. Following the links within the document could result in more queries by the system to determine the meta-information associated with the new document.
- Other complex queries based on the contents of various multimedia objects may be used. For example, “return all documents with video clips depicting research on ...”.

## 5.1 Design Principles

Since multimedia systems have a lot of potential in delivering information to the users, special attention should be given to the actual usability of these systems. A system's usability is determined by how easily and effectively the users can use and communicate with the system. Usability parameters for most systems include ease of use, efficiency, ease of remembering and pleasantness (Nielsen 1990a). Some general and generic design requirements for usable multimedia user interfaces have been identified including simplicity, consistency, engagement, depth and fun to use. In addition to these general design requirements, there are more specific requirements for the News-on-Demand application. The general high-level functions required of the system can be summarized as follows:

- **Browsing/Viewing information:** Users should be able to view multimedia documents by reading text, looking at images, playing video, listening to audio and following links to related information.
- **Searching for information:** Users should be able to search the news using a variety of criteria such as date, author, subject, location, and, most importantly, it's content. The system should provide a fast and easy way for searching and an efficient mechanism for accessing and displaying search results.
- **Customizing the system:** Users should be able to define and modify system settings. Settings should include: document layout, screen layout, window specifications, quality of service parameters and others.
- **Other functions:** These include allowing users to add their own annotations to news documents, providing users with additional navigational aids such as maps and subject indexes, and providing users with a history of the visited documents.

These requirements point to a customizable and easily extensible interface which combines the browsing capability (found in most existing multimedia interfaces) with a querying capability (lacking in many of the same interfaces). Furthermore, the querying capability should be a visual one that merges seamlessly with the browsing facility and satisfies the other general usability requirements of a multimedia user interface.

These requirements suggest a number of design decisions. Our choices vis a vis these decisions can be summarized as follows:

- **Hypermedia:** Hypertext/Hypermedia provides the user with a non-sequential means of freely browsing information according to individual need (Nielsen 1991).  
Not all multimedia systems use hypermedia links (Blattner and Dannenberg 1992). However, in the News-on-Demand application, documents often have links to other related data such as background information, more news coverage, and expert analysis. Therefore, a hypermedia interface is a good design choice as it provides the news readers with an easy and efficient way of accessing and browsing related information.
- **Query mechanism:** A hypertext/hypermedia interface to a multimedia system may not be sufficient to provide all of the accessing mechanisms the user needs to obtain information from the database. In many applications, such as News-on-Demand, users need to search for specific information based on partial knowledge. This must be accomplished more simply and quickly than is possible through the browsing facilities of

hypermedia. Moreover, as the information increases in quantity and complexity, the browsing facility of hypermedia becomes more and more inadequate. According to studies of the usability of hypertext systems, users have often reported that they become disoriented while navigating through hypertext systems and fail to reach points of interest. This phenomenon was reported even when using the most popular commercial hypertext systems (Nielsen 1990b; Nielsen 1991). For the News-on-Demand application, a querying facility that allows users to search and retrieve information directly from the database is a good design choice. The user interface should provide an easy way for performing queries and searches, and examining the results.

- **Input modalities:** A “good” user interface should provide the user with appropriate interaction modes, depending on the application and the types of input needed from the user. We can categorize the modes of interaction into three major categories (Blattner and Dannenberg 1992):
  - Direct manipulation of graphical objects on the screen.
  - The use of natural language.
  - The use of formal languages.

In case of the news-on-demand application, a graphical user interface, with direct manipulation techniques, is sufficient to deal with user input. Typically, users need to click on icons to follow links, choose options from menus and lists, type text, etc.

- **Tight integration with the OBMS:** An important aspect of the visual query facility is its tight integration with the OBMS. Each of the allowed functions has a corresponding ObjectStore query specification. Thus, each user request is translated into an invocation of a query on the database and the translation of the result to the visual form that is requested.
- **Separation of presentation from document storage:** As discussed in Sections 2.3 and 4, we follow the SGML/HyTime principle to separate the document storage from its presentation. We also separate certain presentational features, such as window size, control panels for various displays, etc., from the style sheet and consider these as the *user profile*. This is because styles are associated with individual logical elements, but there are other presentational features that are independent of the types of element being displayed. For example, one user may specify that clicking a video button in a document should start playing the video immediately while another user’s preference may be to open up a video control window with VCR-type controls. These choices are made more frequently than the choices of styles, and may require a different storage model than the one adopted for the document content which assumes no updates to the content.

## 5.2 The Interface

It is not our intention in this paper to describe the full functionality of the interface. Instead, we will highlight the major functions that it supports and focus on the querying capability. The full set of screen shots comprising the user interface (as well as our publications) is available on the World Wide Web at <http://web.cs.ualberta.ca/~database/multimedia.html>. The visual interface we have developed for the news-on-demand multimedia information system provides a number of fundamental functions (Fig. 14):

- initiate a quality-of-service negotiation;
- start a filtering operation in the database;
- perform a search.

In addition, the user can select options from menus in the Application Launcher: (1) The **Implementation** menu provides debugging capabilities by displaying the ObjectStore queries that correspond to the various functions (this issue is discussed further in the following section); (2) The **Document List** menu displays a list of the multimedia documents in the database; (3) The **Tools** menu provides the user with miscellaneous tools which include subject index, history, etc.; (4) The **User Settings** menu allows the user to customize the system settings as part of style sheets.



Fig. 14. Application Launcher

The search and filter operations result in a subset of the news articles being displayed. A user may conduct further searches on these articles. Alternatively, any of the news articles may be opened by clicking on the textual description of that document.

Once a document has been opened, the user can access related material by clicking on an anchor to follow a link. In many multimedia systems, a link can only have a single destination. However, we take the more general approach of allowing multiple destinations where each destination may be a whole multimedia document or a component of a document. The user can navigate between documents by following or returning from links. More importantly, the interface allows the user to perform a search from anywhere within a document and to specify the scope of the search.

Another important feature of the interface is allowing the users to customize the presentation. Customization ranges over a variety of system parameters, such as defining the presentation of anchors within a document, attaching a font format (such as italics, bold, reversed video, etc.) to an emphasized text, and defining different layouts for the same document. Other examples of customization include immediate playing of a video when the icon is clicked versus opening up a control panel (similar to a VCR) to allow explicit activation of the video. The user can also define a default document filter, quality of service parameters, etc. for system startup. All user preferences, for the system and presentation issues, are stored in the database as *user profiles* and/or *style sheets*.

Since our main emphasis in this paper is querying capabilities and their tight integration with the database, we will discuss them in some detail. The full interface functionality is presented elsewhere (El-Medani 1995). Two classes of querying capabilities are provided: filtering documents and searching (both globally and from within a document).

### Filtering Documents

The user can specify the scope of the documents displayed in the document list by using the **Filter** option (Fig. 15). Filtering is a search on document attributes. The result of the filter is a set of documents whose attributes match the ones specified by the user. Filtering of documents can be performed based on subject, country of publication, title, author, date, etc. More attributes can be added by storing them in the database. The set-

tings of any filter can be saved (in the user profile) as the default filter which is used during system startup, but filters can also be applied to the document list at any time.

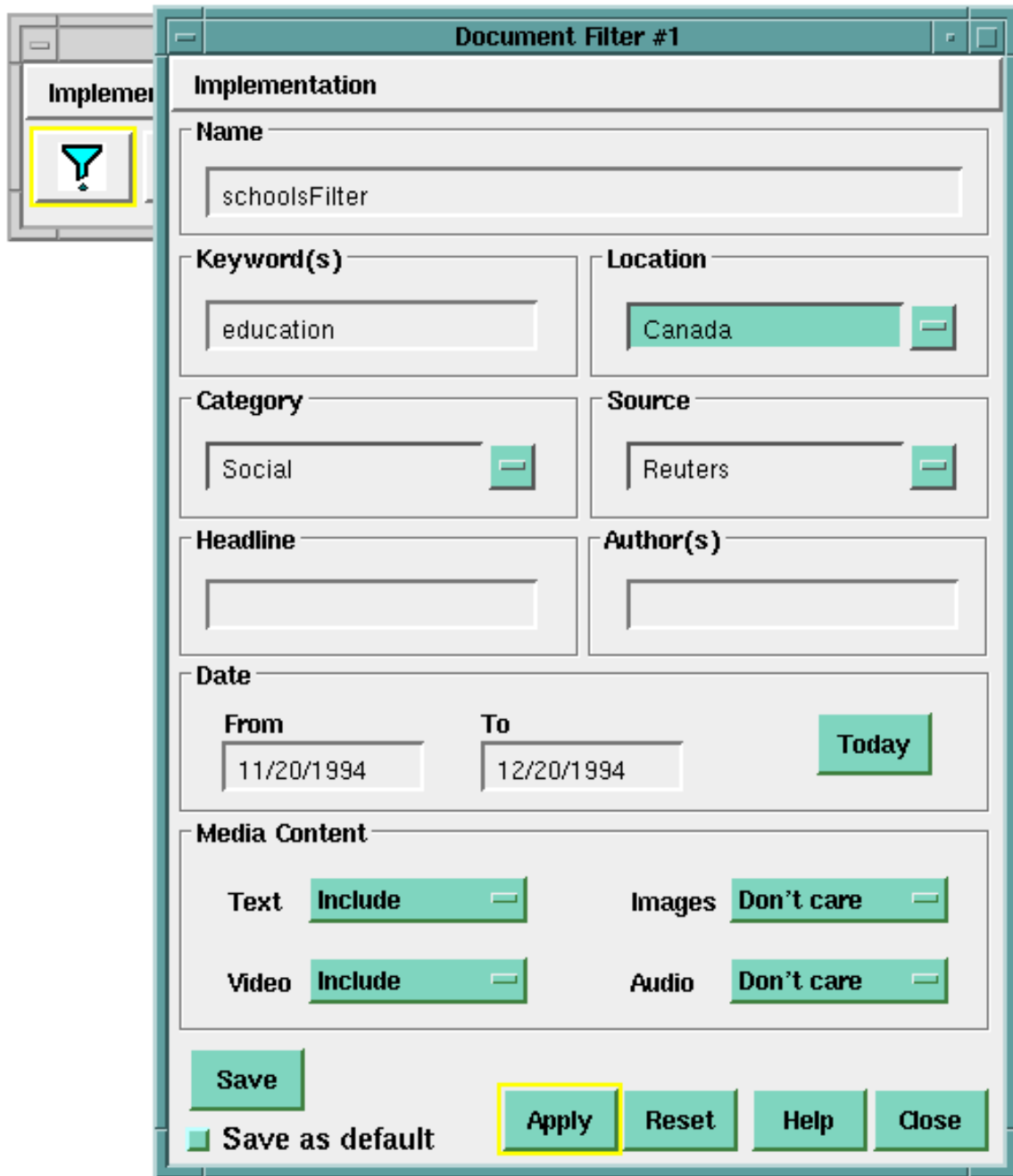


Fig. 15. Filtering Documents

### Searching Documents

The user can search documents for specific information (apart from attributes) by using the **Search** facilities provided by the system. Searches can be performed on the whole database (Global Search), on a document list or from within a document. The user specifies the text to search for (optionally using Boolean combinations of strings), the media types as

well as the scope of the search. The scope of the search can be all the documents in the database, the documents in a list, only the current document, or document directly linked to the current document. Fig. 16 shows a global search on all the documents. For images, audio and video, a keyword search is performed (as indicated before, we have not yet developed content-based indexing and access techniques for these media types). The search facility allows the user to query the database and locate specific information directly as needed. This provides an easier and more efficient way of accessing data than using the navigation facility where exploration can be very time-consuming.

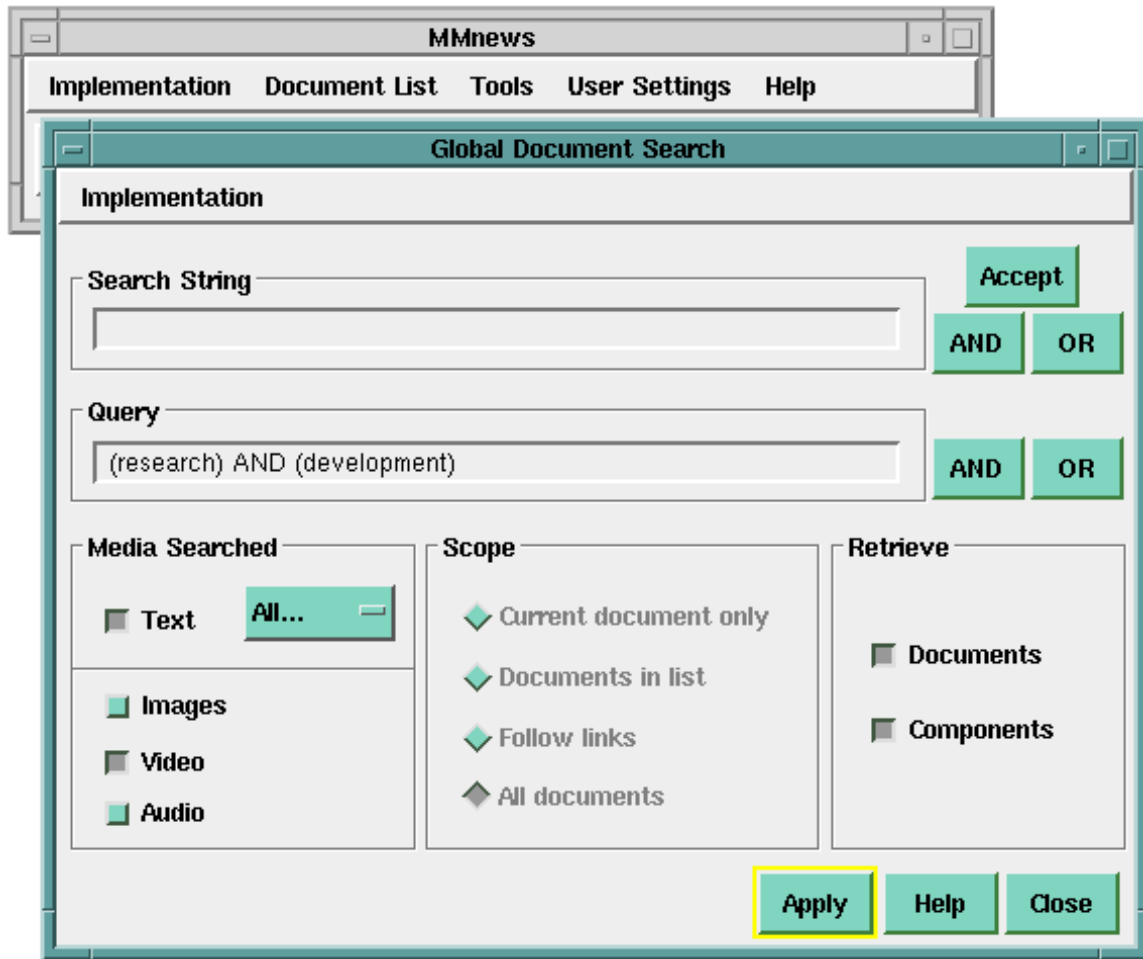


Fig. 16. Global Search

The **Text** submenu of the **Search** option (Fig. 17) allows the user to define the text elements to be searched in the query. A user can choose to search all the text content, or alternatively, can choose to only search specific elements such as the headline, the abstract, paragraphs, quotations, etc.

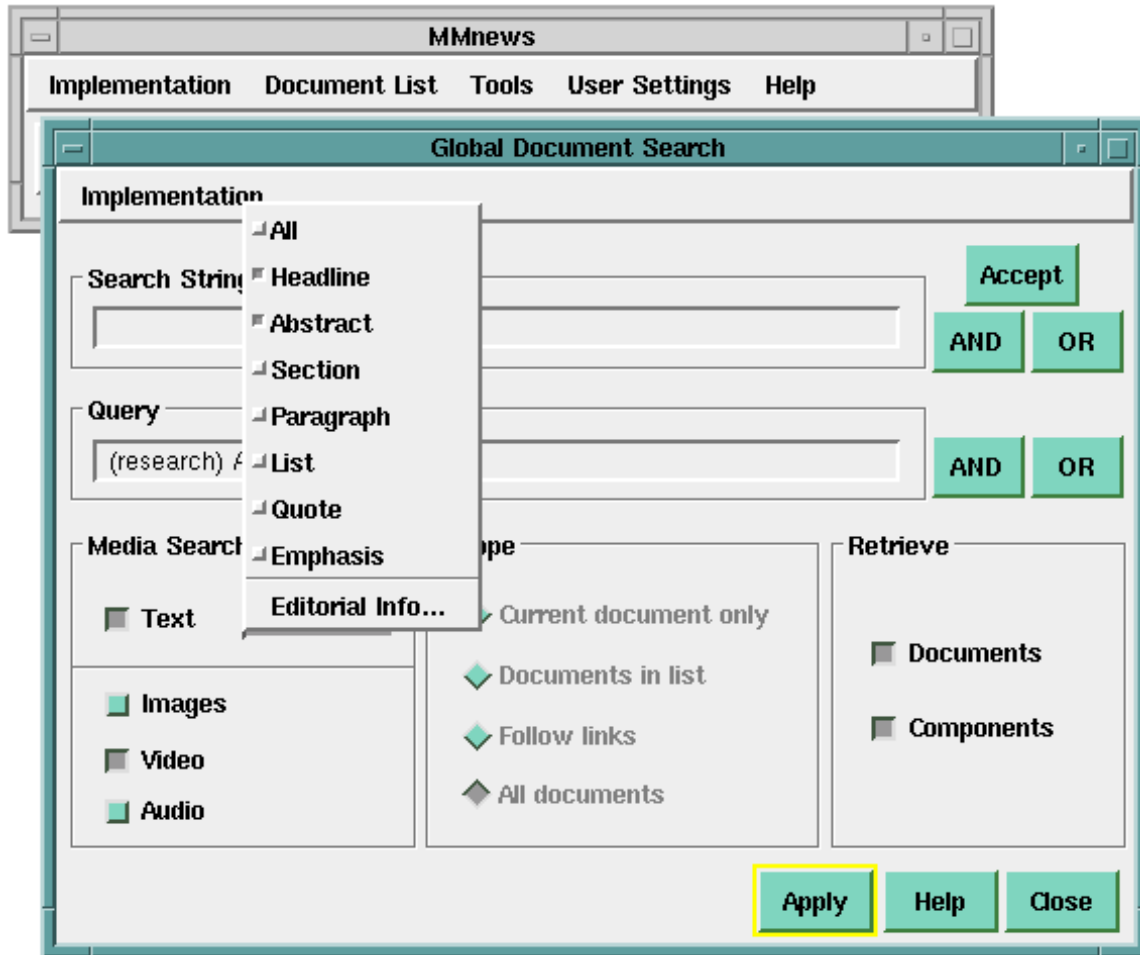


Fig. 17. Text Submenu

The search facility returns a list of objects which match the query (Fig. 18). The objects can be text paragraphs, video clips, images, audio streams or even complete documents. The types of objects returned are specified by the user during the search (it corresponds to the media types checked). For example, if the user selects only video and images as the media types to be searched, the query will only return video and image objects which match the query, but not the associated text or audio. The user can go back and narrow or widen the types of objects returned by re-marking the media types.



Fig. 18. Search Result

Since the search results list consists only of text summaries of objects, there is no need to fetch these objects from the database server. This significantly reduces communication costs since multimedia objects can be very large and expensive to transmit. It is only when the user selects a search result object to view that the object is actually transferred to the client (Fig. 19).

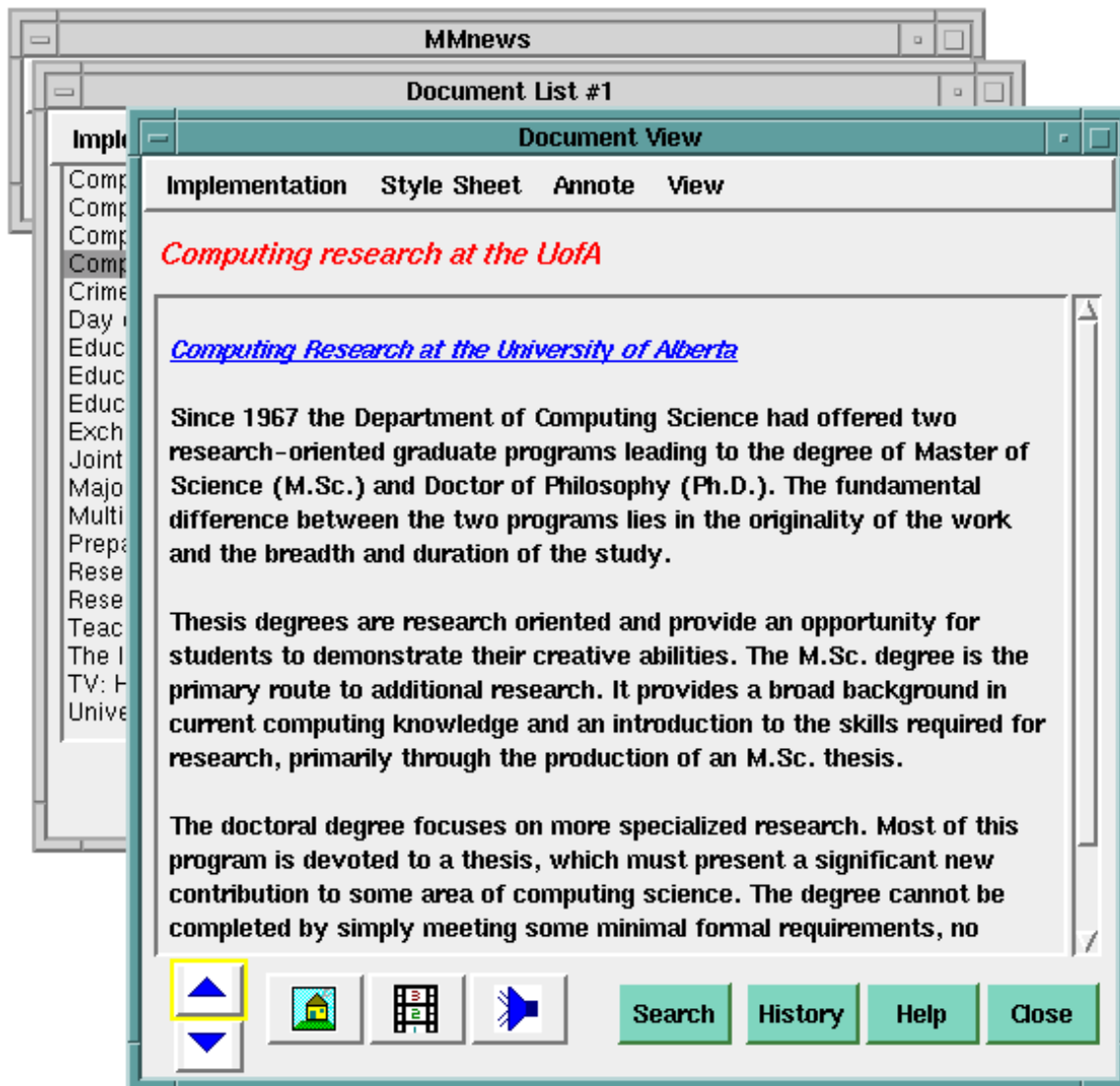


Fig. 19. Document View

## Style Sheets

As indicated earlier, we separate the presentational considerations of multimedia documents from the storage of those documents. These presentational preferences are stored as style sheets and the visual query interface provides facilities for their manipulation. The **Style Sheets** submenu of the Document View (Fig. 20) provides a list of the available style sheets. The user can choose any one of the defined style sheets to be used in displaying the document. Alternatively, the user can define a new style sheet and save it in the database.



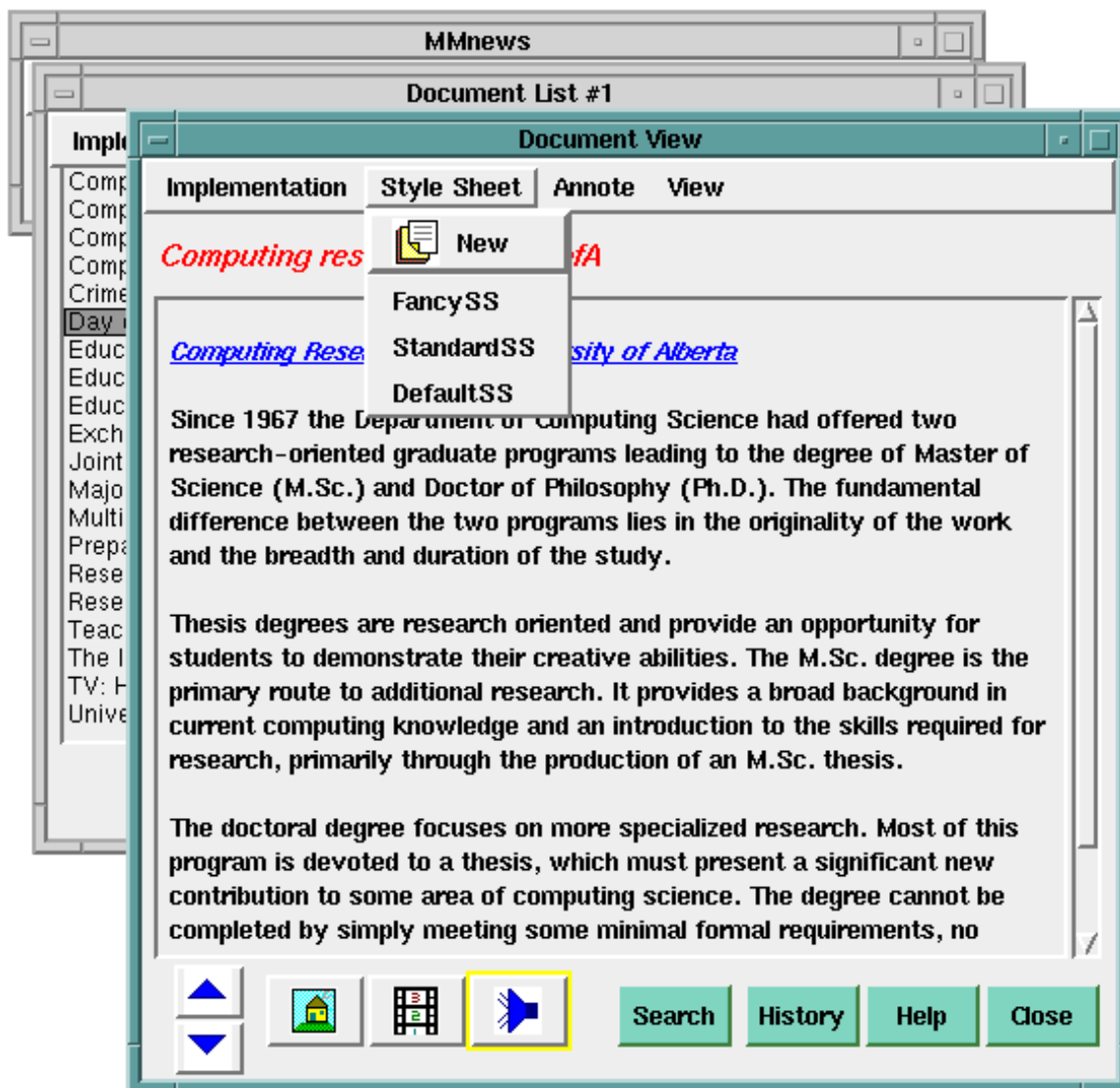


Fig. 20. Available Style Sheets

Any defined style sheet can be updated by the user (Fig. 21). The operations that are allowed include the definition of anchors, attachment of a font format to an emphasis type, choosing a font. In the future, we intend to allow other attributes, related to presentation, to be added to style sheets by the users. This facility is not yet provided since this requires the generation of interface windows and DTD updates for style sheets.

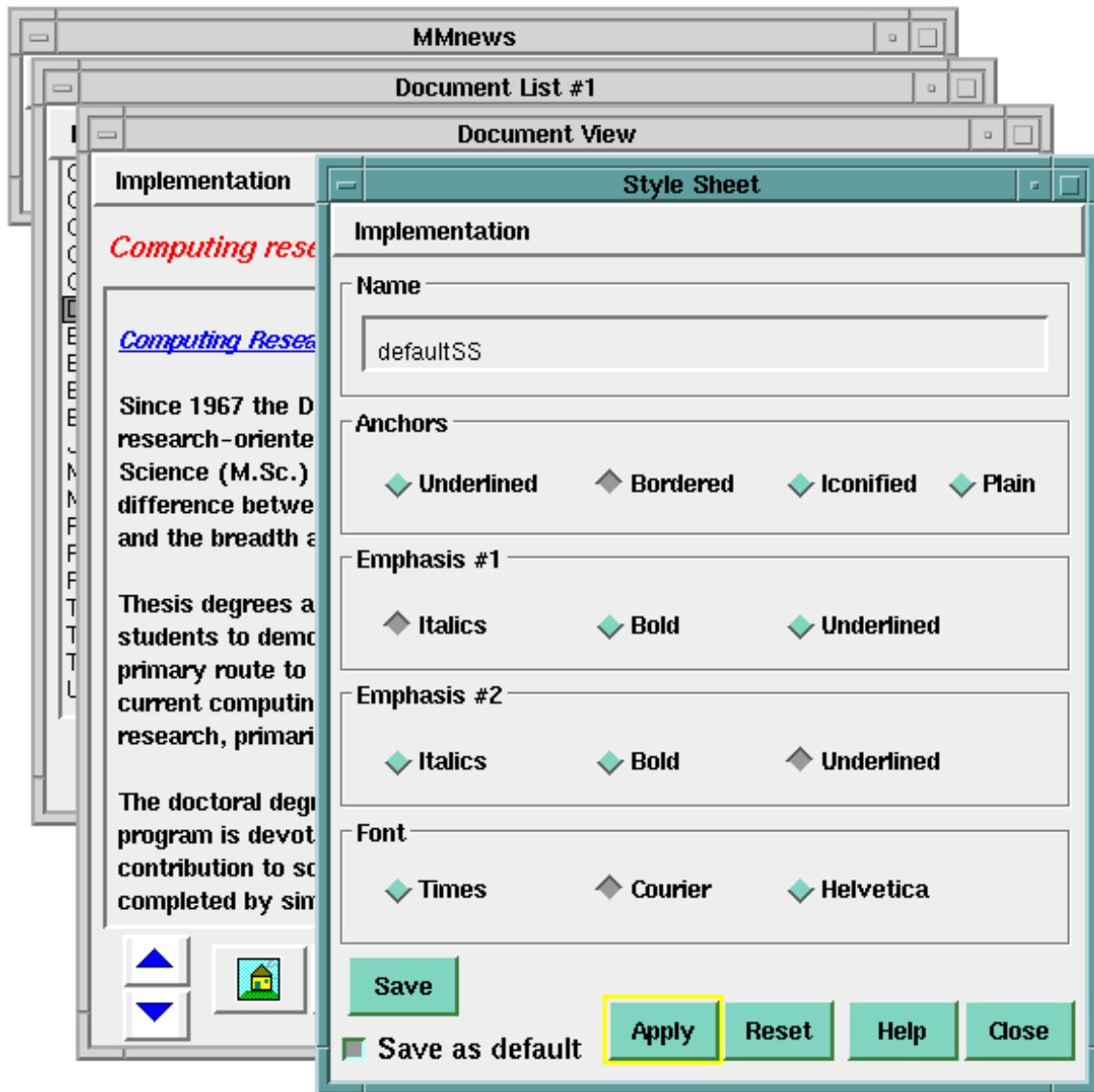


Fig. 21. Style Sheet Settings

### 5.3 Coupling the Interface with the Database

A unique feature of the visual query facility is its tight coupling with the multimedia database. Each time a search is performed or a document component is to be viewed, a database query is issued to the underlying ObjectStore system. The visual query facility, running on a client machine, issues ObjectStore queries to the client which fetches the required objects from the server. The ObjectStore client then returns the matching objects to the interface. It is the responsibility of the visual query interface to manipulate these objects depending on the tasks to be performed. In the remainder of this section, we will describe this coupling by means of an example.

Assume that the user wishes to filter the initial document list that appears in the Document Launcher window (Fig. 14) to include only documents dealing with education in Canada. The user presses the **Filter** button and the Filter window (Fig. 15) is displayed.

After the user fills in the **Subject** and **Country** fields, the **Ok** button is pressed. The visual query interface translates this information into an ObjectStore query as follows.

Assume that the class of multimedia documents is called **article** and is defined as follows in C++ (note that we only include a subset of the attributes here):

```
class article {
public:
    char * title;
    char * country;
    char * date;
    char * source;
    char * subject;
    .....
}
```

The following ObjectStore query will be produced by the visual query interface and will return a collection of all the articles dealing with education in Canada:

```
os_database *news_database;
os_Set<article *> *all_articles;

os_Set<article *>& matching_articles =
    all_articles->query ("article *,"
        "this->subject == 'Education' &&
        this->country == 'Canada'," news_database);
```

ObjectStore queries are always over a single top-level collection, i.e., a collection not embedded in another object. A query returns a subset of the queried collection (`os_Collection::query()`), a single object (single element query; `os_Collection::query_pick()`), or a Boolean (existential query; `os_Collection::exists()`). Queries can be nested (Orenstein et al. 1992). The query constraint (the condition satisfying the query) must use data members or function calls involving a comparison operator. These specifications required by ObjectStore must be taken into account in translating all user actions to equivalent queries; for example, when a document object is fetched for viewing.

## 6. Related Work

The issue of database design for multimedia data has been tackled from the relational as well as the object oriented data modeling perspectives. The design involves (a) defining a model for multimedia documents, and (b) defining models for multimedia data.

The utility of object oriented database systems for hypermedia applications (*vis a vis* relational systems) is highlighted in (Balasubramaniam 1993). Perhaps the earliest object oriented approach is (Kim et al. 1986) which discusses building a multimedia database on top of Orion. In (Blake et al. 1994), the task of incorporating support for text in a relational DBMS is tackled. To enable queries on structured text documents in SGML format, extensions to SQL are proposed. Instances of document categories (defined by different SGML DTD) are fields of a new data type called TEXT. Each TEXT field consists of the contiguous text content of the document along with the parse tree. Elements in the DTD can be part of the query; it is also possible to pose queries about the DTD itself. The EXPAND opera-

tor can be used to convert parts of the parse tree into fields of a relation. Updates to the TEXT field are not handled. Including multimedia data and HyTime elements would not be possible in this design. The approach of storing the text content contiguously and not fragmenting it is similar to our approach. However, we store the locations (or annotations) of the start and end of element instances; the parse tree is implicit in the composition hierarchy (cf. Section 4.4). Other approaches to object oriented models for multimedia data include (Christodoulakis 1991).

A novel object oriented model for a video database is proposed in (Oomoto and Tanaka 1993). The model is *schemaless*, and includes *inheritance by inclusion* as an inheritance mechanism. This means that instances, not types, inherit attributes. Therefore, the hierarchical structure of a video object would be described by a series of derivations, and not by composition.

Querying of SGML documents is also the focus of (Christophides 1994), where extensions of two OBMS query languages are proposed. The paper highlights the issues associated with the object oriented modeling of SGML document structure. In particular, two extensions to the data model of an object oriented database system ( $O_2$ ) are proposed. They are: (a) *ordered tuples*, or the ordering of attributes of a type, and (b) union types. The extensions to the query language are (a) the *contains* predicate to handle querying on strings, (b) *implicit selectors* to handle queries over union types, and (c) two new sorts to query text without exact knowledge of its structure. Types representing unstructured document elements are inherited from basic (atomic) types such as Text and Bitmap. This means that textual document elements are fragments of the text content of the document, which imposes a performance/storage overhead. However, the authors believe that the improved access flexibility makes the new language particularly suitable for extensions to SGML, such as HyTime. Structured document elements have types associated with them; however there are no inheritance relationships shown between them.

Our approach to handling union types is described in Section 4.3. The ordering of attributes is visible through the behavior of the types. We don't handle queries with inexact knowledge of the document structure in our model. Querying on strings is handled by the method `match` of the Text type. Since we do not implement union types in our system, we do not deal with selectors (queries predicated on the discriminant method). The inheritance relationships described in Sections 4.1–4.3, and the composition hierarchies described in Section 4.4 illustrate our approach: every element is an Element, and may have an instance of an atomic type. If the element is a TextElement, then its content is described by its `annotation` attribute.

The design of an OBMS application to handle the storage of SGML documents is described in (Böhm et al. 1993). This design also fragments documents according to the document's SGML type definition, and enables queries on the structure of the document. The paper does not describe the querying facilities, but describes in detail how dynamic DTD handling is implemented by means of meta-classes. The design follows a layered approach by separating out the DTD specific features and DTD independent features into two separate layers of classes<sup>3</sup>. Document type-specific classes are *specializations* of the document type-independent classes. This means that features present in all SGML documents (methods to navigate the document tree for example) can be abstracted out in the classes of the document type-independent layer. Furthermore, there are two meta-classes: *terminal* and *nonterminal*. Classes in the document type-specific layer are instances of either of these

---

<sup>3</sup> There is a third layer, the HyTime layer, which is under development.

meta-classes. The nonterminal class has a method to create a new document-specific class at run time. The content model of the new class can be set using another method. Finally, instances of the document type-specific class can be created at run time using the inherited method `createElem()`. In this manner, DTDs can be dynamically created and inserted into the database.

Our approach has similarities to this design. The supertypes `TextElement`, `Structured`, and `StructuredText` can be said to be document type-independent types. We have not considered dynamic additions of DTDs to the database in our design (although `ObjectStore` has metatypes in its data model and allows the dynamic addition of classes to the `ObjectStore` database schema).

The implementation of a persistent object oriented system for HyTime documents forms part of the paper (Koegel et al. 1994). The database (implemented on `ObjectStore`) forms part of a HyTime engine which is used to process and display hypermedia documents represented using the HyTime standard. This design also fragments the document according to the element types in the DTD. The design is again layered: there is an SGML layer, a HyTime layer, and an application layer.

There are only three classes in the SGML layer: the document class, the element class, and the attribute class. When a document is inserted into the database, an instance of the document class is created, with its fields as the collection of all instances of the elements of the documents. The element instances in turn have references to their attributes which are instances of the attribute class. In the HyTime layer, each architectural form used has a class associated with it. Instances of these AFs get inserted at document insertion time. The application layer has a class for each element type in the DTD. These get instantiated by the application process, which obtains information on them by querying the HyTime and SGML layers. The application then works from this layer. Updates to these objects get propagated down to the appropriate HyTime and/or SGML layers.

The types representing HyTime elements in our design are all subtypes of `HyElement`. This could be considered to be a separate 'HyTime Layer'. However, the application specific HyTime elements (for example `Stext`, `Saudio`) are subtypes, of types representing architectural forms, and not a separate layer. We assume no updates to the instances in the databases.

A document model based on the Office Document Architecture (ODA) is described in (Meghini et al. 1991) and (Bertino et al. 1988). ODA is similar to SGML in that it allows for the specification of the logical structure of the document. In addition, it allows the specification of a *layout* structure, or the presentation information associated with the document. The papers mention object oriented models as candidates to model these structures. They define an additional layer, called the *conceptual structure* which is used to capture the semantics of the components of the logical structure. In (Meghini et al. 1991), it is recognized that support for *multimediality* is required; this is achieved by providing primitive classes for each media type. Querying this document model, and the optimization of such queries forms most of the paper (Bertino et al. 1988).

An object oriented framework for modeling composite multimedia objects (such as multimedia documents) is proposed by the Object Systems Group at the University of Geneva in (Gibbs et al. 1993; Gibbs et al. 1994). The focus is on providing a high level interface for multimedia programming. In particular, (Gibbs et al. 1994) deals with data models for time based media, and (Gibbs et al. 1993) deals with so-called *audio/video (AV) databases*. These databases are collections of digital audio/video data and processes which can

compose and aggregate these data. An AV database, therefore, not only stores data, but is also “involved with the capture, presentation and scheduling of complex objects, managing access and allocation of devices and channel bandwidths, and notifying the application of presentation-related events”. This model performs almost all of the functions of a HyTime engine.

Others have focused on the temporal aspects of multimedia data, and their synchronization. (Hamakawa and Reitmorro 1993) describes a model for composing multimedia objects and the playback of the composite objects. In (Little and Ghafoor 1991), a procedure is presented for the spatial and temporal composition of distributed multimedia objects. *Object Composition Petri-nets* (an augmented model of Petri net with logic of time intervals for Petri net execution) are used to specify the temporal constraints of compound multimedia objects. The destination workstation retrieves the temporal relationships from the server database, evaluates the petri net and derives the playback schedule. (Little and Ghafoor 1993) describes a temporal model to capture the timing relationships between objects in composite multimedia objects, and maps it to a relational database. This model forms a basis for a hierarchical data model and for temporal access control algorithms to allow VCR-like capabilities. They show how it can be mapped to a relational database and derive the playback algorithm.

HyTime also adopts an interval based approach to modeling timing relationships, and establishes a composition hierarchy (Figure 13) using the document structuring feature of SGML. All intervals are defined on a single time axis for the whole document. In our implementation, this representation is mapped to an instance of a Time Flow Graph (Lamont and Georganas 1994) (a playback schedule)<sup>4</sup> used by the synchronization component of the distributed multimedia system to synchronize data streams retrieved from the servers.

Many multimedia information systems provide a browsing-based user interface, but have limited querying capabilities. Perhaps, the most popular interface of this type is Mosaic for accessing the World Wide Web. Along the same lines, a model for hypertext-based information retrieval is presented in (Lucarella 1990). In (Williams 1991), a hypermedia system for on-line technical documentation is described. It provides a browsing facility and supports limited keyword search. There are many other hypermedia systems which provide elaborate browsing facilities but they have limited querying capabilities (Grønbaek and Trigg 1994; Haan et al. 1992; Plaisan 1991). The system described in (Frei and Schauble 1991) supports storage and retrieval, as well as browsing of multimedia information. These functions are achieved through hyperlinks between related information and a searching facility. Queries are handled using a functional database query language (FQL\*) which is extended to include elementary similarity functions to allow for imprecise queries.

## 7. Conclusions and Future Work

In this paper we describe an object-oriented multimedia DBMS design for a news-on-demand application. The focus of the work reported here is the development of a type system that supports multimedia documents and the development of a visual query interface for accessing a multimedia database. There are three characterizing features of our type system design: (1) the central use of DBMS technology, (2) the reliance on object-oriented sys-

---

<sup>4</sup> Type `Av_fcs`, which models a Finite Coordinate Space has a method `GetTimeSchedule` which returns a data structure representing the binary temporal relationships between the events in the FCS. From this data structure, the synchronization module derives the Time Flow Graph.

tems, and (3) strict adherence to international standards. The database is designed to accommodate actual multimedia objects as well as meta-information about them. The database schema consists of an object type system which follows the SGML/HyTime standard for document preparation. The query facility provides two major capabilities: *querying* and *browsing*. Querying capabilities are necessary to allow users to directly and efficiently retrieve information from the database. Browsing capabilities allow users to easily explore and obtain information that is directly related to query results. Our visual interface combines these capabilities by concentrating on two areas of technology: query mechanisms and hypertext/hypermedia systems. It enables users to access the multimedia database without having to type complicated ObjectStore queries.

This database is implemented on top of ObjectStore running under AIX. The implementation language is x1C, which is IBM's implementation of C++ for the AIX environment.

In the long-run, we are developing an extensible OBMS that has inherent support for multimedia formation systems. We intend to use our own system, called TIGUKAT<sup>5</sup> (Özsu et al. 1995), to eventually replace ObjectStore. We may not be able to achieve the same performance, but there will be opportunities to expand on the functionality and investigate the feasibility of various issues. It is difficult, if not impossible, to investigate all of the issues related to multimedia DBMS design by building a layer on top of a closed system such as ObjectStore. TIGUKAT is currently being prototyped at the Laboratory for Database Systems Research of the University of Alberta. It has a purely behavioral object model where the users interact with the system by applying behaviors to objects. In this way, full abstraction of modeled entities is accomplished since users do not have to differentiate between attributes and methods.

In the short-term, we have started to investigate four issues: (1) making the type system richer, (2) building front-ends to the multimedia DBMS, (3) content-based indexing and querying of multimedia objects, and (4) the development of application specific query languages and primitives, and the optimization of these queries. Our type system design implements a DTD definition for multimedia news documents. We are investigating ways of storing DTDs as objects and using meta-types to define new DTDs.

Secondly, our work, so far, has concentrated on the right-hand-side of Fig. 1. Consequently, a multimedia document is currently entered into the database by creating the object instances according to the types defined in the database. We are trying to couple our multimedia DBMS with an SGML compiler which will be retrofitted to instantiate object instances at the "code generation" step of compilation. Later on, we will add HyTime capability to this front-end. In this fashion, documents marked-up according to the SGML/HyTime standard can be automatically inserted into the database.

Thirdly, we are investigating content-based indexing of multimedia objects. At this point, the only way to query images, for example, is to define a number of attributes and search on the values of these attributes. It is important to provide the facility to pose queries that refer to the content of the images rather than the attributes defined on them and be able to deal with such queries. The initial step in providing this facility is to work on indexing of image contents. We intend to incorporate these into an image storage system that will be

---

<sup>5</sup> TIGUKAT (tee-goo-kat) is a term in the language of Canadian Inuit people meaning "objects." The Canadian Inuits, commonly known as Eskimos, are native to Canada with an ancestry originating in the Arctic regions of the country.

coupled with TIGUKAT.

Finally, we are studying access primitives and query languages for multimedia information systems. Particularly, we are interested in the design of application-specific query languages by which application designers can access multimedia objects. The development of the appropriate language features for multimedia systems is often cited as one of the most important issues in the development of multimedia applications. These language features will be incorporated into the TIGUKAT query model and the query language developed on top of it.

## References

Böhm K, Aberer K, Hürer C (1993) Extending the scope of document handling: The design of an OODBMS application framework for SGML document storage. Technical Report P-93-24, GMD-IPSI, Germany

Balasubramaniam V (1993) State of the art review on hypermedia issues and applications. Internal document, Graduate School of Management, Rutgers University, Newark, New Jersey

Blake GE, Consens MP, Kilpeläinen P, Larson PA, Snider T, Tompa FW (1994) Text/relational database management systems: Harmonizing SQL and SGML. Proceedings of the First Intl. Conf. Appl. of Databases, pp 267-280

Blattner MM, Dannenberg R.B (ed) (1992) Multimedia Interface Design. ACM Press, New York

Bertino E, Rabitti F, Gibbs S (1988) Query processing in a multimedia document system. ACM Trans. Office Information Systems 6(1):1-41

Christophides V, Abiteboul S, Cluet S, Scholl M (1994) From structured documents to novel query facilities. Proceedings of the ACM SIGMOD International Conference Management of Data, pp 313-324

Christodoulakis S, Ailamaki N, Fragonikolakis M, Kapetanakis Y, Koveos L (1991) An object-oriented architecture for multimedia information systems. Q. Bull. of IEEE Tech. Comm. on Data Eng. 14(3): 4-15

S. J. DeRose SJ, Durand DG (1994) Making Hypermedia Work – A User's Guide to HyTime. Kluwer Publishers, Boston

Dimitrova N, Golshani F (1992) EVA: A query language for multimedia information systems. Proceedings of the International Workshop on Multimedia Information Systems, pp 1-20

Dogac A, Özsu MT, Biliris A, Selis T (1994) Advances in Object-Oriented Database Systems. Springer-Verlag, Berlin

El-Medani G (1995) Design and implementation of a hypertext user interface for a multimedia kernel. M.Sc. Thesis, Department of Computing Science, University of Alberta, Edmonton, Canada

Frei HP, Schauble P (1991) Designing a hypermedia information system. Proceedings of



- the DEXA 91. Database and Expert Systems Applications Conference, pp 449-454
- Gibbs S, Breiteneder C, Tsichritzis D (1993) Audio/video databases: An object-oriented approach. Proceedings of the 9th International Conference on Data Engineering, pp 381-390
- Gibbs S, Breiteneder C, Tsichritzis, D (1994) Data modeling of time-based media. Proceedings of the ACM International Conference on Management of Data, pp 91-102
- Goldfarb CF (1990) The SGML Handbook, Oxford University Press, Oxford
- Grønbæk K, Trigg RH (1994) Design issues for a dexter-based hypermedia system. Communications of the ACM 37(2):40-49
- Haan BJ, Kahn P, Riley VA, Coombs JH, Meyrowitz NK (1992) IRIS hypermedia services. Communications of the ACM 35(1):36-51
- Hafid A, Bibal A, Bochmann, Gv, Burdin T, Dssouli, R, Gecsei J, Kerhevé B, Vu Q (1994) On news-on-demand service implementation. Publication #928, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Canada
- Hamakawa R, Reikmoto J (1993) Object composition and playback models for handling multimedia data. Proceedings of the First ACM International Conference Multimedia, pp 273-281
- ISO (1986) Information Processing – Text and Office Information Systems – Standard Generalized Markup Language (ISO 8879). International Standards Organization
- ISO (1989) Office Document Architecture (ODA) and Interchange Format (ISO 8613). International Standards Organization
- ISO (1992) Hypermedia/Time-based Structuring Language: HyTime (ISO 10744). International Standards Organization
- Koegel JF, Rutledge LW, Rutledge JL, Keskin C (1993) HyOctane: A HyTime engine for an MMIS. Proceedings of the First ACM International Conference Multimedia, pp 129-136
- Little TDC, Ghafoor A (1991) Spatio-temporal composition of distributed multimedia objects for value added networks. Computer 24(10): 42-50
- Little TDC, Ghafoor A (1993) Interval-based conceptual models for time-dependent multimedia data. IEEE Trans. Know. and Data Eng. 5(4):551-663
- Lamont L, Georganas ND, Synchronization architecture and protocols for a multimedia news service application. Proceedings of the IEEE International Multimedia Computing and Systems Conference, pp 3-8
- Lamb C, Landis G, Orenstein J, Weinreb D (1991) The ObjectStore database system. Communications of the ACM 34(10): 50-63
- Lucarella D (1990) A model for hypertext based information retrieval. Hypertext: Concepts, Systems and Applications. Proceedings of the European Conference on Hypertext, pp 81-94

- Meghini C, Rabitti F, Thanos C (1991) Conceptual modeling of multimedia documents. *Computer* 24(10): 23–30
- Nielsen J (1990a) *Hypertext & Hypermedia*. Academic Press, New York
- Nielsen J (1990b) The art of navigating through hypertext. *Communications of the ACM* 33(3):296-310
- Nielsen (1991) Usability considerations in introducing hypertext. In: Brown H (ed) *Hypermedia/Hypertext And Object-Oriented Databases*. Chapman & Hall, London, pp 3–17
- Neufeld G, Makaroff D, Hutchinson N, (1994) The design of a file server for scalable VBR media, Technical Report, Department of Computer Science, University of British Columbia, Vancouver, Canada
- Orenstein J, Haradhvala S, Margulies B, Sakahara D (1992) Query processing in the ObjectStore database system. *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pp 403-412
- Oomoto E, Tanaka K (1993) OVID: Design and implementation of a video-object database system. *IEEE Trans. Knowledge and Data Man.* 5(4):629–643
- Özsu MT, Peters RJ, Szafron D, Irani B, Lipka A, MunozA (1995) TIGUKAT: A uniform behavioral objectbase management, *VLDB Journal*, in press
- Özsu MT, Valduriez P (1991) *Principles of Distributed Database Systems*, Prentice-Hall, Englewood Cliffs
- Plaisant C (1991) An overview of Hyperties, its user interface and data model. In: Brown H (ed) *Hypermedia/Hypertext And Object-Oriented Databases*. Chapman & Hall, pp 17-31
- Vittal C, Özsu MT, Szafron D, El-Medani G (1994) The logical design of a multimedia database for a News-on-Demand application. Technical Report TR94-16, Department of Computing Science, University of Alberta, Edmonton, Canada
- Williams I (1991) Hypermedia for multi-user technical documentation. In: Brown H (ed) *Hypermedia/Hypertext And Object-Oriented Databases*. Chapman & Hall, pp 17-31
- Woelk D, Kim W, Luther W (1986) An object-oriented approach to multimedia databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp 311–325

# APPENDIX 1

## DOCUMENT TYPE DECLARATION FOR MULTIMEDIA NEWS ARTICLES

```
DOCTYPE article SYSTEM article.dtd [  
  
<!-- HyTime Modules Used -->  
<?HyTime support base>  
<?HyTime support measure>  
<?HyTime support sched manyaxes=3>  
<?HyTime support hyperlinks>  
  
<! -- Non-HyTime Notations used -->  
<!NOTATION virspace PUBLIC -- virtual space unit (vsu)--  
  +//ISO/IEC 10744//NOTATION Virtual Measurement Unit//EN>  
  
<! -- Document Structure -->  
<!ELEMENT article - - (frontmatter, async, sync)>  
<!ELEMENT frontmatter - - (edinfo, hdline, subhdline, abs-p)>  
<!ELEMENT edinfo - - (loc & date & source & author+ &  
  keywords & subject)  
<!ELEMENT (loc|source|subject) - - (#PCDATA)>  
<!ELEMENT (hdline|subhdline) - -  
  (emph1|emph2|quote|link|#PCDATA)+>  
<!ELEMENT date - - (#PCDATA)>  
<!ELEMENT (author|keywords) - - (#PCDATA)>  
<!ELEMENT abs-p - - paragraph>  
<!ELEMENT async - - (section|Fig.|link)*>  
<!ELEMENT section - - (title?, (paragraph|list)*)>  
<!ELEMENT title - - (#PCDATA) >  
<!ELEMENT paragraph  
  - - (emph1|emph2|list|Fig.|link|quote|#PCDATA)*>  
<!ELEMENT (emph1|emph2|quote) - - (#PCDATA) >  
<!ELEMENT list - - (title?, listitem+)>  
<!ELEMENT listitem - - (paragraph)*>  
<!ELEMENT link - - (emph1|emph2|quote|Fig.|#PCDATA)+>  
<!ELEMENT Fig. - - (figcaption?) >  
<!ELEMENT figcaption - - (#PCDATA) >  
<!ELEMENT sync - - (audio-visual+)>  
<!ELEMENT audio-visual - - (x, y, time, av-fcs, av-extlist+)>  
<!ELEMENT (x|y|time) - - EMPTY>  
<!ELEMENT av-fcs - - (av-evsched+)>  
<!ELEMENT av-evsched - - (audio*, video*, stext*)>  
<!ELEMENT (audio|video|stext) - - EMPTY >  
<!ELEMENT av-extlist - - (xdimspec, ydimspec,tdimspec)>  
<!ELEMENT (xdimspec|ydimspec|tdimspec) - - (axes-marklist)>  
<!ELEMENT axes-marklist - - (#PCDATA)>
```

```

<!ATTLIST article
  id          ID      #REQUIRED
  HyTime      NAME    #FIXED HyDoc>

<!ATTLIST quote
  source      CDATA  #IMPLIED>

<!ATTLIST author
  designation CDATA  #IMPLIED>

<!ATTLIST Fig.
  filename    CDATA#REQUIRED
  format      CDATA#REQUIRED>

<!ATTLIST (x|y|time)
  HyTime      NAME    #FIXED axis
  id          ID      #IMPLIED
  axismeas    CDATA#FIXED virspace
  axismdu     CDATA#FIXED
  axisdim     CDATA#FIXED virspace>

<!ATTLIST link
  HyTime      NAME    #FIXED ilink
  id          ID      #REQUIRED
  linkends    IDREFS  #IMPLIED>

<!ATTLIST av-fcs
  HyTime      NAME    #FIXED fcs
  id          ID      #REQUIRED
  axisdefs    CDATA#FIXED    x y time>

<!ATTLIST av-evsched
  HyTime      NAME    evsched
  id          ID      #REQUIRED
  axisord     CDATA#FIXED    x y time
  basegran    CDATA#FIXED    vsu vsu vsu>

<!ATTLIST (audio|video)
  HyTime      NAME    #FIXED event
  id          ID      #REQUIRED
  filename    CDATA#REQUIRED
  format      CDATA#REQUIRED>

<!ATTLIST stext
  HyTime      NAME    #FIXED event
  id          ID      #REQUIRED
  filename    CDATA#REQUIRED >

<!ATTLIST av-extlist
  HyTime      NAME    #FIXED extlist
  id          ID      #REQUIRED>

<!ATTLIST av-dimspec
  HyTime      NAME    #FIXED dimspect
  id          ID      #REQUIRED>

<!ATTLIST axes-marklist
  HyTime      NAME    #FIXED marklist
  id          ID      #REQUIRED>]

```

## APPENDIX 2

### DTD FOR STYLE SHEETS

```
<!DOCTYPE style-sheet SYSTEM style-sheet.dtd [  
<!ELEMENT style-sheet - - (cat-name, rule+)>  
<!-- Document category name, e.g. article -->  
<!ELEMENT cat-name - - (#PCDATA)>  
<!-- rule which maps element (source) to style (spec)-->  
<!ELEMENT rule - - (source, spec*)>  
<!ELEMENT source - - (#PCDATA)>  
<!-- each presentation attribute has a value -->  
<!ELEMENT spec - - (pres-attr, value)>  
<!ELEMENT pres-attr - - (#PCDATA)>  
<!ELEMENT value- - (#PCDATA)>  
<!-- No Attributes for any of the style sheet elements -->>]
```