Database management support for a news-on-demand application

M. Tamer Özsu, Duane Szafron, Ghada El-Medani, Sherine El-Medani,
Paul Iglinski, Manuela Schoene, Chiradeep Vittal

Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

ABSTRACT

We describe the type system component of a database management system that supports a multimedia news-on-demand application. The type system is an object-oriented one that represents document structure according to the SGML and HyTime standards. End-users access the news database by using a visual query interface. We also describe current work to generalize the database type system to accommodate arbitrary SGML/HyTime compliant multimedia documents. Such a generalized type system would support a broad range of multimedia applications.

Keywords: multimedia, news-on-demand, multimedia database, SGML, HyTime.

1. INTRODUCTION

News-on-Demand is an application which provides subscribers with access to multimedia news articles that are inserted into a distributed database by news providers. Commercial news gathering/compiling organizations such as wire services, television networks, and newspapers are examples of news providers. 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.

Currently, there are a number of news-on-demand projects, but most of these systems do not use database technology to store or manage the documents. However, there are a number of distinct advantages to using database management systems (DBMS) for this kind of application. First, multimedia systems can benefit from standard DBMS services like data independence (data abstraction), application neutrality (openness), controlled multi-user access (concurrency control), fault tolerance (transactions, recovery), and access control. Second, traditional file systems force the user to format files for multimedia objects and to manage large amounts of data. Third, databases can also represent meta-data like document structure and spatio-temporal relationships along with the data. This approach can provide a uniform query interface that is based on the structure of documents and the relationships between documents as well as on the content of the documents. Finally, multimedia applications, including the news-on-demand application, are generally distributed, requiring multiple servers to satisfy their storage requirements. The well-developed distributed DBMS technology[1] can be used to efficiently manage data distribution.

In this paper we describe the type system of a DBMS that supports a distributed news-on-demand application. This DBMS is part of the larger Broadband Services Project that is being conducted by six institutions with support from the Canadian Institute for Telecommunications Research. The objective of the project is to develop a software infrastructure and to define an API that is suitable for a broad range of broadband distributed multimedia applications. The scope of the project is currently being expanded from presentation-only applications, such as news-on-demand, to collaborative applications. The multimedia DBMS component of this project covers a broad spectrum of activities from the design and implementation of an appropriate database type system (schema) to the development of application-specific query models and languages that support content-based access to multimedia objects. This paper focuses on one major component of the first phase of the project, the design of the database schema for the news-on-demand application. Figure 1 shows this component in context of the entire application environment. This paper also outlines our current research on (a) extending the database schema to support additional applications, and (b) coupling the DBMS with an SGML parser to enable automatic insertion of documents into the database.

The unique features of our work are the following:

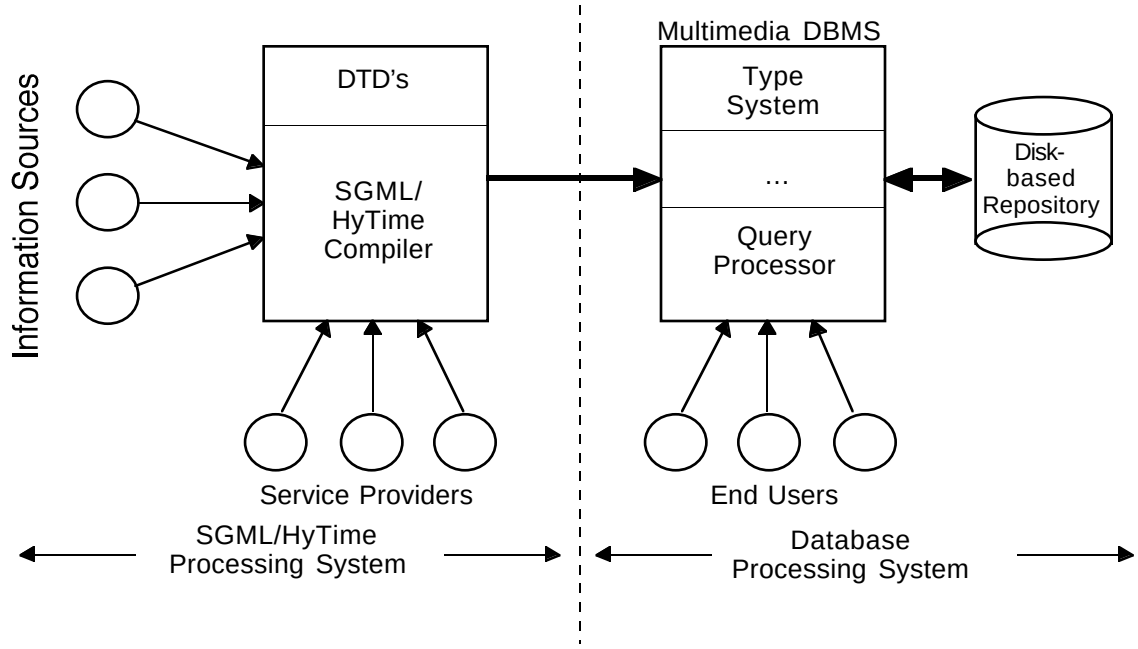

Figure 1. Processing environment

1. An integrated database stores not only directory information about the location of multimedia objects, but also the objects themselves. This is quite different from other multimedia information systems where the database is constrained to store only descriptive information and all of the multimedia objects are stored in ordinary files. This system stores non-continuous media (text and images) as native objects in the database and uses a continuous media file server[2], that is being developed at the University of British Columbia, as its underlying storage system for continuous media (audio and video).

2. The ObjectStore™ object-oriented DBMS[3] is used as the base layer of the multimedia database. The choice of an object-oriented database instead of a more common relational system was made for a number of reasons. Relational DBMSs are suitable for representing meta-information, but are inadequate for modeling the complex structure of multimedia documents. The "binary large objects" (BLOBs) that are supported in some relational systems are not sufficient to model multimedia entities. Although images can be stored as BLOBs, it is not possible for the system to interpret the internal structure of BLOBs. For example, the system could not identify all BLOBs that contain a particular sub-image. Furthermore, multimedia systems require extensible database schemas while relational systems support only a fixed set of data types (integer, real, character, date, etc.).

3. The type system strictly adheres to the *Standard Generalized Markup Language (SGML)* and the *Hypermedia/Time-Based Structural Language HyTime* standards[4,5]. These are ISO standards (numbers 8879 and 10744) that are sufficiently rich to support the target application, and are gaining widespread popularity. For example, HTML has become the de-facto document standard for the World Wide Web and it is a popular application of SGML. SGML mostly deals with textual documents whereas HyTime adds support for hypermedia and synchronized documents (e.g., links and video). There are many successful projects that approach database schema design in an ad hoc fashion. However, in an area such as multimedia information systems, it is important to follow international standards so that the multitude of applications and tools that follow these standards (e.g., authoring tools, browsers) can all be used with the multimedia database.

The rest of this paper is organized as follows. Section 2 presents the database schema for the news-on-demand application. It is a summary of work that has already been completed. Section 3 describes current research to generalize the type system to support any SGML/HyTime documents. On-going work to facilitate automatic insertion of documents into the database is described in Section 4. Section 5 highlights the long-term directions that are being followed.

2. DATABASE SCHEMA DESIGN

In the object-oriented world, database schema design corresponds to the design of a type system[*]. In this context, three fundamental issues must be considered. First, the different basic media components of the document (i.e., text, image, audio, and video) need to be modeled. Second, the structure of the multimedia news documents must be represented. Third, the spatial and temporal relationships between multimedia objects have to be captured and stored in the database. There is a fourth issue related to the storage of descriptive control information that is required by other components of the multimedia application. However, we do not consider the fourth issue in this paper. In the remainder of this section we summarize our approach to addressing the first three issues. More details can be found in other[**] publications[6,7,8].

2.1 Modeling of Basic Multimedia Objects

Since ObjectStore does not provide native support for basic multimedia data other than text (strings), the type system defines these data types and refers to them as *atomic types* (Figure 2).

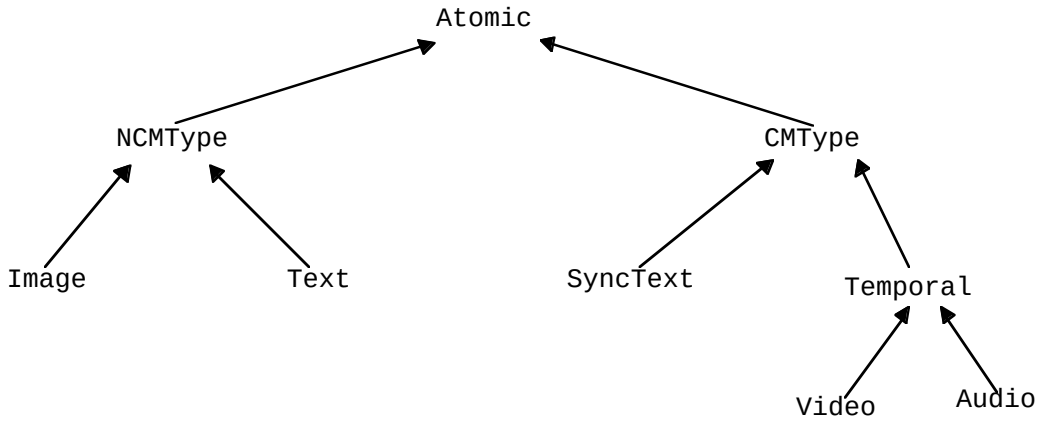

Figure 2. Atomic type hierarchy

All atomic types are defined as subtypes of an abstract type, `Atomic`. There are two abstract subtypes of atomic media, one for non-continuous media (`NCMType`) such as text and images and another for continuous media (`CMType`) such as audio and video. All attributes and methods which are common to both kinds of media are promoted to the `Atomic` type.

The abstract supertype, `NCMType` defines a non-continuous media attribute called `content`, which is an array of bytes, but it is further subtyped by the concrete media types, `Text` and `Image`. The `Text` subtype has additional methods: `match` which implements a pattern matching algorithm, and `substring` which returns a portion of the text object given a start index and an end index. The `Image` type has additional attributes such as the width, height and colors of the image. Both these types have attributes for the quality of service (QoS) parameters specific to the media they model. The `Image` type can be further subtyped to reflect the different storage formats possible, such as JPEG, TIFF or GIF.

Note that the actual data corresponding to objects of type `CMType` (and its subtypes) are stored in the continuous media file server, rather than the multimedia DBMS. Thus, for the present implementation, database objects of these types contain only meta-information. Nevertheless, a subtyping scheme has been created on the `CMType` side of the type hierarchy. The `Temporal` supertype of video and audio is defined to promote the common `duration` attribute. The `Video` type can be subtyped to handle different storage formats like MPEG and Motion-JPEG. Synchronized text (`SyncText`), a form of closed captioning, is not subtyped from `Text`, since it is not stored in the database. Instead, it is stored on the file system with other continuous media. As a consequence, the methods `match` and `substring` cannot be applied to synchronized text media.

---

[*] In this paper we use the terms “type” and “class” interchangeably. Some object-oriented systems make a distinction between types and classes. However, ObjectStore follows the C++ convention of using “classes” for both concepts. Thus, any reference to “defining a type” corresponds to “defining a C++ class” in ObjectStore.

[**] Our publications are available on-line on the World Wide Web at URL http://web.cs.ualberta.ca/~database/

2.2 Modeling of Document Structure

As stated earlier, we follow the SGML standard for representing document structure. SGML formally specifies this structure by defining element types (e.g., paragraph, figure) and the relationships between them in a *Document Type Declaration* (DTD). SGML does not pre-specify the nature of these elements, or the structure of the composition hierarchy that contains them. Instead, a document designer specifies a different DTD for each category of document being designed. For example, a single "Book" DTD hierarchically composed of chapter, section, paragraph and word elements might serve as the template for many instances of book. In our case, we wrote a DTD for multimedia news articles. However, since we are interested in a full-blown object-oriented DBMS for storing and querying news articles, we also designed a type system that conforms to this multimedia news DTD.

A DTD specifies *element types*, the hierarchical relationships between element types, and *attributes* associated with them. Attributes contain "hidden" information that is not part of the document content. For example, a DTD designer may define a Figure element with a fileName attribute that is a String representing the name of the file that contains a bit-map image for the figure. The filename itself is not part of the content of the document.

The type system must represent all of the information contained in the composition hierarchy and in the attributes. We designed a type hierarchy for the elements in the news DTD which is rooted at the abstract `Element` type (Figure 3). All elements maintain a reference to their parent element in the document instance hierarchy, so that the hierarchy can be navigated starting from any element. This type is responsible for implementing these references. `Element` is subtyped by three more specialized abstract types `TextElement, Structured` and `HyElement`.

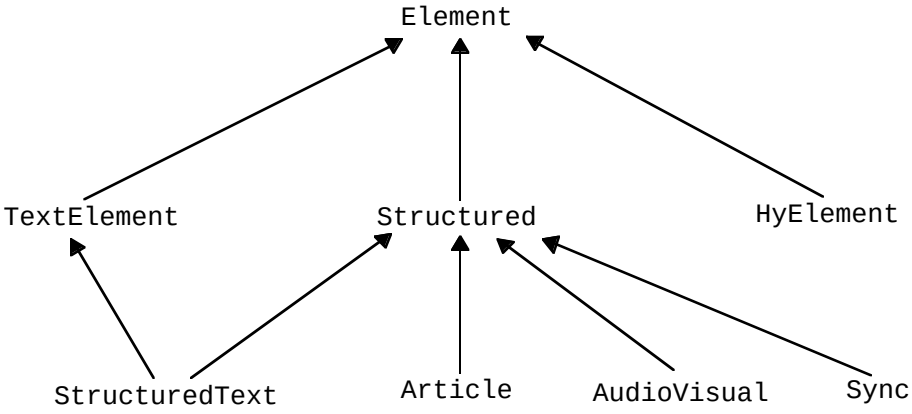

Figure 3. Element type hierarchy

In the DTD for news documents, we divide the document into asynchronous and synchronous components. This reflects the fact that continuous media with synchronization constraints (synchronous) need to be handled by HyTime conforming element types, and other SGML element types are adequate to deal with text and image data (asynchronous). The supertype `HyElement` encompasses all the HyTime elements used in the DTD. HyTime types are discussed in Section 2.3.

Capturing the document structure as a set of types raises the question of the storage model for the text part of the multimedia documents. The standard approach is to store pieces of text with the elements that logically contain them. For example, a Book may contain a list of Chapters, where each chapter contains a list of Sections. Each Section may contain a list of paragraphs. Each paragraph may ultimately contain a String. Alternately, a finer granularity occurs if each Paragraph contains a list of Words and each Word contains a String.

Although this standard approach is elegant and straightforward, it is very inefficient for two major reasons. First, the number of individual objects that must be fetched from disk is large. Second, searching for Strings that span element (e.g., Word or Paragraph) boundaries is difficult. Our storage model for the textual components of a document is novel. In our type system, the entire text string of each document is stored in a single object and compositional elements are represented as annotations on the text object. Each annotation stores a reference to the text object, as well as start and end indexes.

The `TextElement` type implements the protocols for manipulating these annotations, including the `getString` method that returns an appropriate sub-string representation of any annotation. The type `Quote` is an example of a direct concrete subclass of TextElement since each quote simply contains a reference to the document's text string, a start index and an end index.

The type `Structured` is a supertype for elements in the DTD with complex content models. By complex, we mean any element whose contents are not either empty or a single String (a simple String is referred to as `#PCDATA` in SGML). That is, structured elements maintain references to child elements. Therefore, the type `Structured` defines the method `getNth` to return the n^{th} child element. The type `Article` is an example of a direct concrete subtype of `Structured`. Each `Article` contains an instance of `FrontMatter`, an instance of `ASync` and an instance of `Sync`.

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`, `Figure`, `Frontmatter`, etc. whose components are themselves either instances of `TextElement` or `StructuredText`. For example, instances of type `Figure` contain both a `FigureCaption` and an `Image`.

2.3 Modeling of Spatio-Temporal Relationships

The representation of spatio-temporal relationships between multimedia objects is an important consideration in designing a multimedia database. This information is required by the synchronization routines to plan the retrieval and synchronized presentation of multimedia objects. For example, both the audio track and closed captions of a video must be synchronized with the video images. Our representation is compliant with the HyTime standard, which addresses the more general issue of presentation. The HyTime philosophy is to separate presentation information from the content of multimedia documents. For example, the tags in a document may specify that a particular segment of the document is emphasized without restricting how this emphasized segment should be rendered. The presentation information specifies presentation preferences. For example, the presentation preferences may specify that emphasis should be rendered as bold, as italics or in an alternate color, without affecting the document contents.

In following the HyTime philosophy, we completely separate the presentation of a document from its content. This has two implications. First, the users' presentation preferences must be stored and accessed when necessary. This is accomplished using individualized *style sheets* which are stored in the database as objects. This aspect of the DBMS is discussed elsewhere[8]. The second, and arguably more important, consideration is to represent the spatio-temporal relationships in accordance with the HyTime standard.

HyTime defines a number of *architectural forms* to deal with various hypermedia concepts. One of these architectural forms is the *finite coordinate space* (FCS) which HyTime uses for modeling spatio-temporal relationships. A finite coordinate space is a set of axes of finite dimensions. All measurements are associated with the *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. We define an FCS of three dimensions: *x* and *y* to represent spatial dimensions and *time* to model the temporal dimension. A set of ranges along the various axes defining the FCS form an *extent* which corresponds to an *event*. An *event schedule* consists of one or more events. Event schedules, therefore, are used to represent temporal relationships among various multimedia objects.

Within this context, our model of spatio-temporal is a set of type definitions that correspond to the relevant HyTime concepts. The type hierarchy is depicted in Figure 4. The type `HyElement` is the supertype for all HyTime elements in the type system. Its immediate subtypes model 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 a string representing the name of the architectural form. This models the assumption that every HyTime element can be linked its architectural form.

3. GENERALIZING THE TYPE SYSTEM

The type system described in the previous section is specific to the news-on-demand application since it is based on the DTD for news documents. The multimedia database will only be able to support other applications if this type system is generalize. Within the context of SGML/HyTime, this means that the database type system has to be able to reflect multiple DTDs. This requires the system to analyze new DTDs and automatically generate the types that correspond to the elements they define. We are currently modifying our system to accomplish this task. In addition, the DTD is stored as an object in the database so that users can run queries like "Find all DTDs in which a 'paragraph' element is defined."

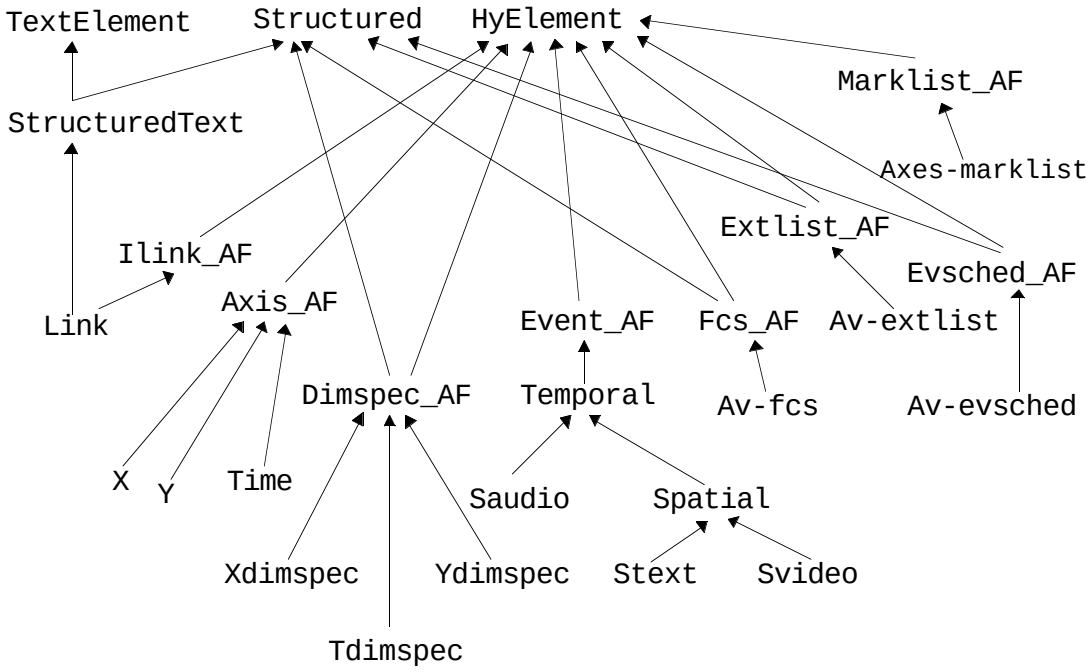

Figure 4. Type hierarchy for HyTime elements

The general architecture of the extended system is depicted in Figure 5. We have a meta-DTD that describes a grammar for defining DTDs and a DTD Parser parses each DTD according to this grammar. While parsing the DTD, an object is created for each valid SGML element defined in the DTD. Each DTD element object contains information about the element, such as its name, attribute list and context model. If the DTD is valid, a Type Generator is used to automatically generate C++ code that defines a new ObjectStore type for each element in the DTD. For example, if a Book DTD is parsed, objects representing: Title, AuthorList, Chapter, Section, Paragraph, Index etc. would be created. If the DTD is valid then each of these objects would generate C++ code that would define its own ObjectStore type.

There are two important problems that need to be addressed in this process. Both problems are abstraction problems that can reduce the complexity of the multimedia type system and therefore reduce maintenance time and errors. First, if two or more DTD elements in the same DTD definition share common features, then this feature should be automatically extracted and promoted to an abstract superclass. For example, in the news-on-demand type system, the two types, `Video` and `Audio` both share a common duration attribute, so the abstract supertype `Temporal` was created to promote this feature. However, this factoring must be done automatically. If the feature is a common component, this is straightforward. Otherwise, the problem is harder to solve.

Second, common element definitions across different DTD definitions should be represented by a common type in the type system. However, there is no easy solution to this problem since it leads to the well-known semantic heterogeneity problem that has been studied extensively within the multidatabase community. Briefly, the problem is one of being able to determine whether two elements are *semantically* equivalent. This problem has also been studied in the programming languages field as well where there are many different definitions for type equivalence[9]. For example, two types are name equivalent if they have the same name. However, this would not be a good definition of type equivalence in our model since two different DTDs might use the same name to describe different elements. For example, a `Signature` in a Thesis DTD may be different than a `Signature` in a Symphony DTD. Similarly, programming languages define two types to be structurally equivalent if the components recursively have the same names and types. This may also lead to faulty equivalencies. For example, `FigureCaption` and a `Title` are structurally equivalent since they each have a single component that is a String. However, they are semantically different and this difference may only become clear in the context of what composite objects can contain them. Since this is not a trivial problem, we have chosen to give up some abstraction in favor of a semantically "safe" type system.

However, this does not mean that we have completely abandoned type re-use across DTDs. We reuse both the atomic types such as Audio, Image and Text (see Figure 2) as well as the high-level abstract supertypes such as TextElement,

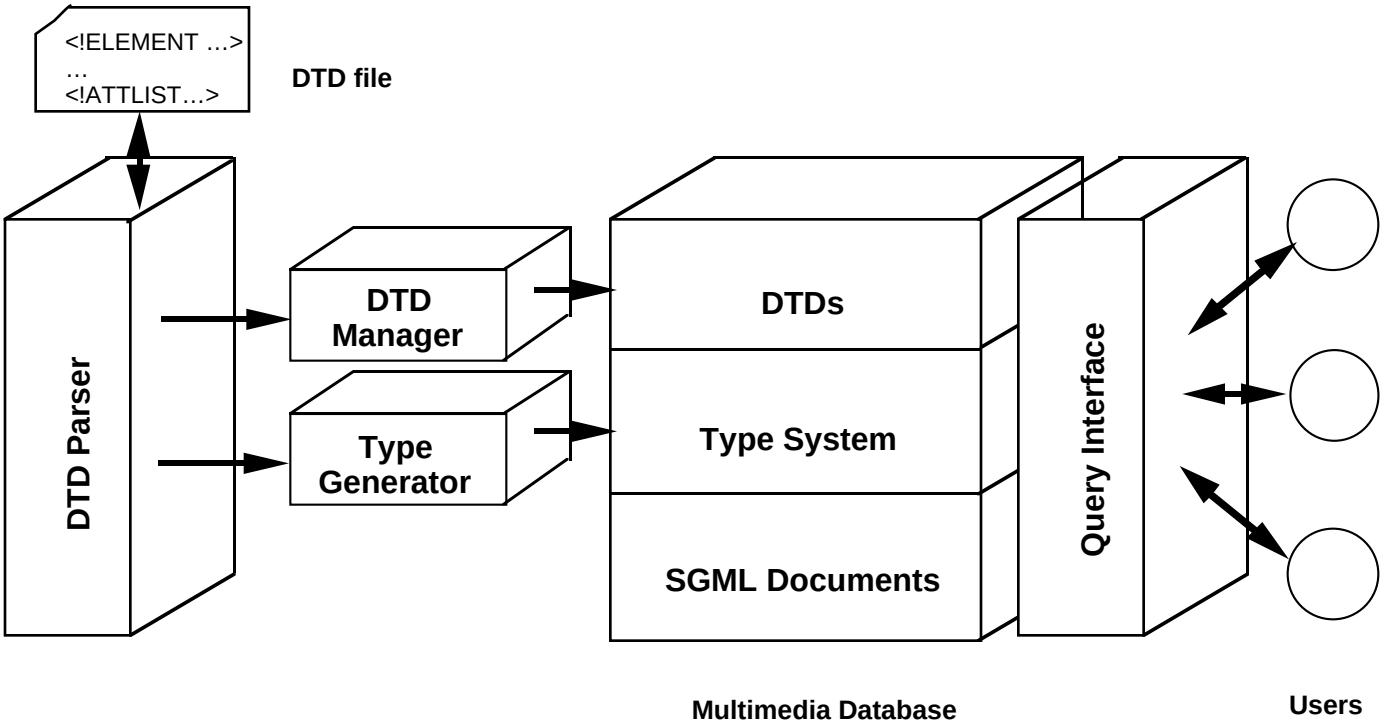

Figure 5. Architecture for handling multiple DTDs

Structured and HyElement (see Figure 3). These types are safe to re-use because they have well defined semantics and appear across many document types. For the rest of the elements in a given DTD, we create new types. Name conflicts between elements in different DTDs are resolved automatically by using the DTD name as prefix during type creation (e.g. article_section, book_section).

This approach actually has one major advantage over re-using arbitrary type definitions across multiple DTDs. That is, new element types are inserted into the database without costly schema evolution. For example, assume an existing DTD defined a type T1 and a new DTD defined the semantically equivalent type T2. Also assume that type T2 had a supertype T3, schema evolution would be necessary for the existing type T1 since T3 would have to be added as a supertype

The DTD Manager in Figure 5 takes the DTD file as input and stores the DTD as an object in the database that can be used for parsing documents and other purposes. This is done after type creation. As soon as a DTD is stored in the database, SGML documents of that type can be inserted. A DTD object contains the name of the DTD, the content of the DTD as a character string and other descriptive information such as the count of the number of documents that conform to this DTD.

4. AUTOMATING DOCUMENT ENTRY

One of the serious shortcomings of our current implementation is the unavailability of authoring tools for the insertion of documents into the database. We have implemented facilities for querying the database[8] once the documents are inserted in it, but no tools exist to automatically insert documents. Part of our current work concentrates on coupling the multimedia database with a retrofitted SGML parser. SGML documents can then be created using existing authoring tools[***] and automatically inserted into the database.

[***] Besides the availability of a number of authoring tools that generate SGML marked-up files, Microsoft has announced that one of the upcoming versions of its popular word processor "Microsoft Word" will be able to save documents in marked-up files. Coupling the parser that we are retrofitting with these authoring tools would not be difficult.

The general architecture for this coupling is depicted in Figure 6. The SGML Parser accepts an SGML Document Instance from the Authoring Tool, validates it, and forms a parse tree. The Instance Generator traverses the parse tree and instantiates the appropriate objects in the database corresponding to the elements in the document. These are persistent objects stored in the database that can be accessed using the query interface.

The parser is based on a freeware application called *nsgmls* developed by James Clark. The parser is being modified to incorporate the following changes:

1. The DTD used for parsing the document instance is fetched from the multimedia database.
2. The output of the parser is passed to the Instance Generator as a parse tree instead of producing parsed text output.
3. The parser does not produce any output unless the document is error-free.

5. CONCLUSIONS

In this paper we describe several components of our DBMS for multimedia applications. Most of our effort so far has concentrated on the news-on-demand application. The resulting type system described in Section 2 is quite specific to the requirements of this and it has been fully implemented as class extensions to ObjectStore on IBM RS6000 machines operating under AIX. In addition to the database schema described here, we have implemented a visual querying facility that enables easy access to documents in the database. This work is described elsewhere[6,8].

We are in the process of generalizing our DBMS so that it can support other multimedia applications which follow the SGML/HyTime standard for describing multimedia documents. Our aim is to make the database usable not only in presentational applications such as news-on-demand, but also in conversational applications. Our approach is outlined in Section 3. This is on-going work and more results will be described in later papers.

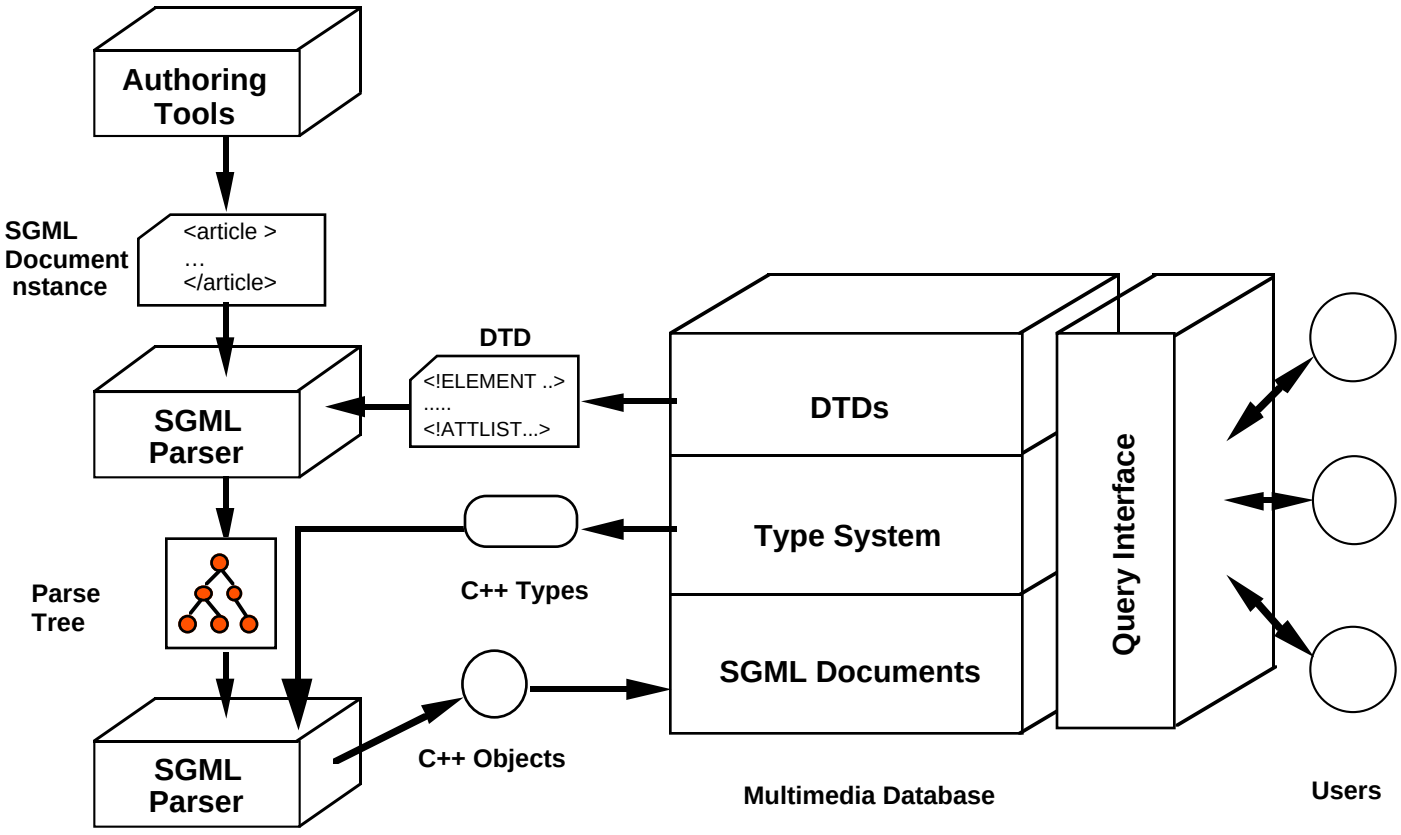

Figure 6. Architecture for automatic document entry

We are also coupling the DBMS with authoring tools to support database document insertion. The approach, outlined in Section 4, uses existing tools that have been developed for SGML and retrofits them to work with our DBMS. This illustrates one major advantage of following a standard such as SGML/HyTime. It enables us to leverage the work of others who have developed a wide range of tools that comply with these standards.

Our long term plan involves three goals. First, to build multi-media-specific query models and languages that are easier to use and more efficient than general query models and languages. These advantages would follow from a common type system that is optimized for queries that are common across multi-media applications, but may not be common in general non-multimedia applications. Second, to support content-based indexing and access to images. For example, the user should be able to ask for "all images that have dogs in them". Third, better coupling of the database with the continuous media file server. Each of these is a long term project whose results will be reported separately.

We did not provide an extensive comparison of our work with related projects. There are quite a number of efforts to design multimedia DBMSs and we have surveyed them elsewhere[6]. The two fundamentally unique aspects of our approach are the adherence to international standards, and the design of an integrated database that not only captures descriptive data about multimedia objects, but also stores the objects themselves.

6. ACKNOWLEDGMENTS

This research is supported by a grant from the Canadian Institute for Telecommunications Research (CITR) which is one of the Networks of Centres of Excellence funded by the Government of Canada.

7. REFERENCES

1. M.T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, Prentice Hall, Englewood Cliffs, NJ, 1991.

2. G. Neufeld, D. Makaroff, and N. Hutchinson. The design of a file server for scalable VBR media, Technical Report, University of British Columbia, Department of Computer Science, December 1994.

3. C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. "The ObjectStore database system," *Communications of the ACM*, Vol. 34, No. 10, pp. 50-63, October 1991.

4. International Standards Organization. *Information Processing – Text and Office Information Systems – Standard Generalized Markup Language (ISO 8879)*, 1986.

5. International Standards Organization. *Hypermedia/Time-based Structuring Language: HyTime (ISO 10744)*, 1992.

6. M.T. Özsu, D. Szafron, G. El-Medani, and C. Vittal, "An object-oriented multimedia database system for a news-on-demand application", *Multimedia Systems*, Vol. 3, No. 5/6, 1995 (in press).

7. C. Vittal, An object-oriented multimedia database system for a news-on-demand application, M.Sc. Thesis, University of Alberta, Spring 1995. Available as Technical Report TR95-14.

8. G. El-Medani, A visual query facility for multimedia databases, M.Sc. Thesis, University of Alberta, Fall 1995. Available as Technical Report TR95-18.

9. K. C. Louden, *Programming Languages, Principles and Practice*, PWS Publishing Company, Boston, MA, 1993.