Multimedia Data Management Research at the University of Alberta

M. Tamer Özsu
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
Tel: +1-403-492-2860
Fax: +1-403-492-1071
Email: ozsu@cs.ualberta.ca
http://web.cs.ualberta.ca/~ozsu

ABSTRACT

We describe the multimedia systems reseach that is currently being carried out within the Laboratory for Database Systems Research of the University of Alberta. The projects focus on data management issues such as high-level query capability, application independent storage and management of data, versioning, meta data management, etc.

1. INTRODUCTION

There are a number of on-going multimedia research projects within the Laboratory of Database Management Systems at the University of Alberta. The research projects range from the design of an object-oriented multimedia database system to high-level multimedia query languages and content-based indexing and access of images. In this paper, we provide an overview of these projects; further information can be found on our World Wide Web page located at http://web.cs.ualberta.ca/~database/.

The multimedia project currently involves four faculty members (Drs. M. Tamer Özsu, Xiaobo Li, Ling Liu and Duane Szafron), one research associate (Paul Iglinski), one laboratory manager (Randal Kornelson), two Ph.D. students (Zhong Li and Youping Niu) and four Master's students (Sherine El-Medani, Cheng Lin, Adriana Manas, and Manuela Schoene). Some of the projects are part of a Broadband Services Project that is being conducted by six institutions with support from the Canadian Institute for Telecommunications Research (CITR) — one of the Networks of Centres of Excellence funded by the Government of Canada. Other components are funded by a strategic grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

2. AN OBJECT-0RIENTED SGML/HYTIME DATABASE SYSTEM

The use of database technology in multimedia information systems is quite restricted. Where they have been used, database management systems (DBMSs) have stored meta data about multimedia objects with the objects themselves stored in regular files. However, there are a number of distinct advantages to using DBMSs 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 such as 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 can be used to efficiently manage data distribution.

The above observations motivate our work in developing a multimedia DBMS. The overall architecture of the system is depicted in Figure 1. The most important part of this design is the type system. In this context, there are three fundamental issues to deal with. 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, meta data about the multimedia objects and applications have to be captured and stored in the database. These meta data include the spatial and temporal relationships between multimedia objects, descriptive control information that is required by other components of the multimedia application, and the like.

We use the ObjectStore™ object-oriented DBMS as the underlying storage system of the multimedia database. 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*. Currently, these *atomic types* store meta data about all media types as well as (about?)the text objects themselves. Images are stored in an external file, and the audio and video media are stored in a continuous media server. The multimedia DBMS, however, provides a uniform interface to all of these repositories. In the future, we will couple the multimedia DBMS and these repositories more closely.

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 Declara-*

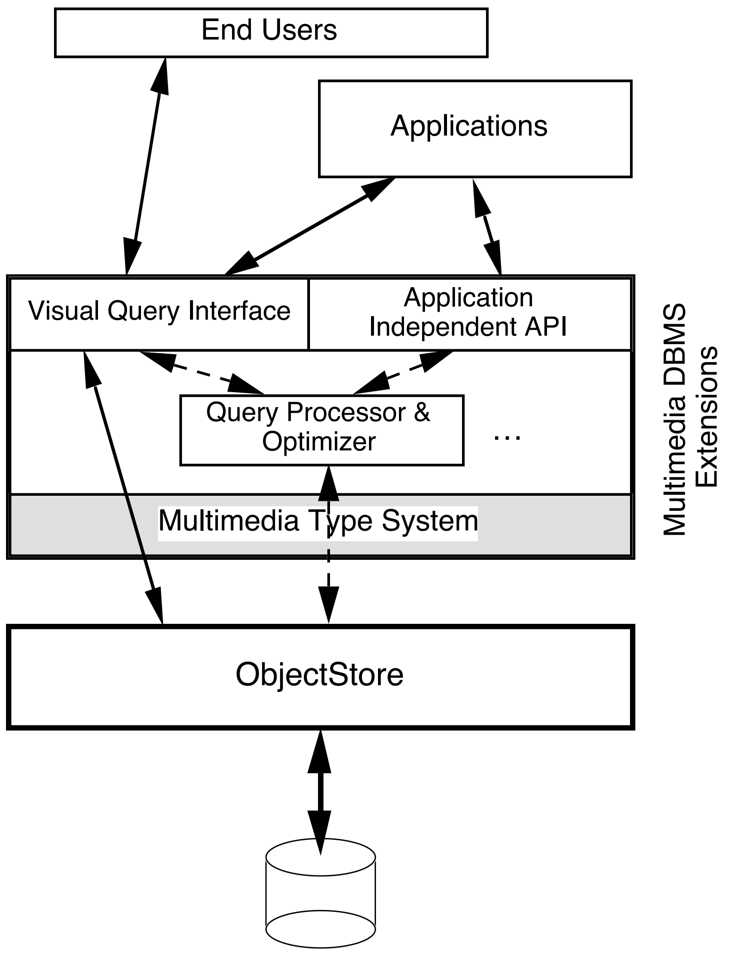
End Users
Applications
Visual Query Interface
Application Independent API
Query Processor & Optimizer
...
Multimedia Type System
Multimedia DBMS Extensions
ObjectStore

Figure 1. Overall System Architecture

tion (DTD). SGML does not pre-specify the nature of these elements, nor 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. The initial design of our system was based on a particular DTD for a target multimedia news application. We are now in the process of generalizing the system, as described in the next section.

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. The HyTime philosophy is to separate presentation information from the content of multimedia documents. 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. 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 relationsip is a set of type definitions that correspond to the relevant HyTime concepts.

One fundamental requirement of a multimedia DBMS for SGML/HyTime documents is that it should be able to handle multiple DTDs and support the creation of types that are induced by these DTDs. This is essential if the multimedia DBMS is to support a variety of applications. This requires the system to analyze new DTDs and automatically generate the types that correspond to the elements they define. We are completing a project to facilitate this process. We store the DTD as an object in the database so that users can run queries like "Find all DTDs in which a 'paragraph' element is defined."

The general architecture of the extended system is depicted in Figure 2. 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 of these 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, where there are many different definitions for type equivalence. 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.

This does not mean, however, that we have completely abandoned type re-use across DTDs. We re-use the atomic types such as Audio, Image and Text as well as the high-level abstract supertypes such as TextElement, Structured and HyElement. 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

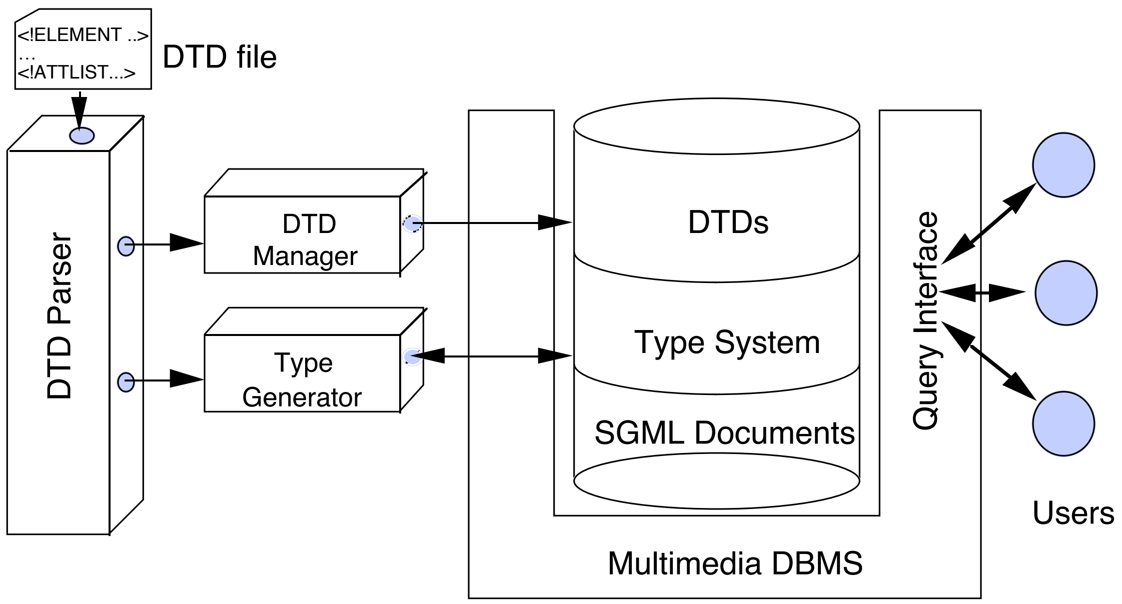

Figure 2. Architecture of the General System

DTDs. That is, new element types are inserted into the database without costly schema evolution. For example, assume that 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.

3. AUTOMATING DOCUMENT ENTRY

One of the serious shortcomings of many multimedia DBMS projects is the unavailability of tools for the insertion of documents into the database. Many systems have facilities for querying the database once the documents are inserted in it, but no tools exist to automatically insert documents. This is generally considered to be outside the scope of database work. One of our projects 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.

The general architecture for this coupling is depicted in Figure 3. 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.

4. DISTRIBUTED IMAGE DATABASE SYSTEM

This project addresses the development of data management technology for images. With the increasing popularity of multimedia information systems, managing image data efficiently is becoming more important (vital?). Most of the systems store images in flat files; it is our contention that the use of database management technology for this purpose would present new opportunities for system development. Since many of the existing multimedia systems store images in files, it is necessary to develop an interoperable system that can accommodate images stored in various systems. This project, therefore, develops a distributed, interoperable image database management system.

The identifying characteristics of our project are (a) object-oriented approach to image data management, (b) use of image processing and indexing techniques for efficient querying and access to image databases, and (c) interoperability among various image storage systems. The result will be fundamental contributions to the solution of a number of fundamental issues of database management and image processing — including the efficiency of image compression mechanisms, the support for progressive and selective image transmission, the scalability of existing data integration approaches, the flexibility of existing content-based image querying strategies, and the transparent accessibility of images from multiple, disparate source data repositories.

One of the issues that we have focused on as part of this project is the content-based indexing and storage of images. This is the initial first step in providing a declarative query capability whereby users can pose queries that

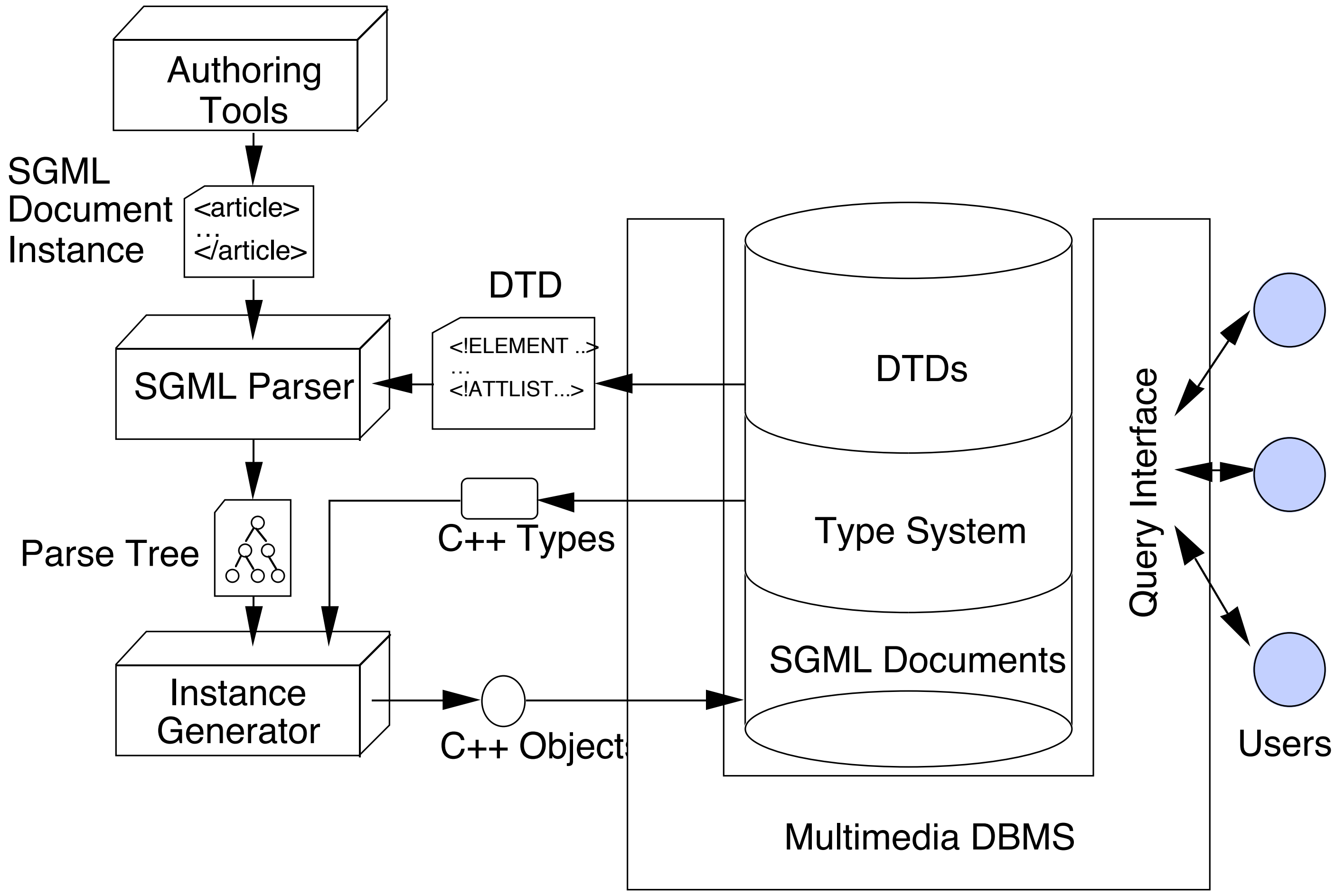

Figure 3. Architecture for automatic document entry

refer to the content of images rather than the attributes defined on them. In other words, the system should be able to support searches that are based on the contents of multimedia objects. We are studying this problem as part of the work on multimedia query system. The more important issue here is to be able to index multidatabase objects based on their contents.

Our work is currently focused on the development of an indexing scheme based on 2-dimensional (2-D) strings. 2-D string representation is one of the more efficient schemes for modeling spatial relationships in images. After preprocessing by image processing and pattern recognition techniques, the original image can be represented by a 2-D string according to its projection along the x- and y-axis, respectively. The 2-D string contains symbols, representing the objects from the image, and spatial relations among symbols (objects). The problem of content-based retrieval of images then becomes the 2-D string subsequence matching.

We are also starting to work on the interoperability issues related to distributed image databases. Most of the image repositories are not database systems. Furthermore, each of these repositories may be coded in a different fashion (JPEG, GIF, etc). Thus, providing general DBMS functionality to access these images requires the management of different repository systems. We are taking an object-oriented approach to interoparability and looking at building wrappers around the various repositories to facilitate uniform access.