A Framework for Multimedia Database Systems *

Vincent Oria, Paul J. Iglinski and M. Tamer Özsu
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1

Abstract

This paper discusses a general framework for multimedia DBMSs. We use our two ongoing multimedia projects (a SGML/HyTime DBMS and an image DBMS) as examples to support the general framework. Although multimedia data (text, sound, image, video and animation) are different in terms of their processing, similarities exist in the ways to handle these data. They all are big with complex structures and semantics. The semantics have to be extracted, stored, and indexed to allow content-based queries.

Résumé

Cet article s'inspire des 2 projets multimédia en cours dans notre laboratoire pour définir une architecture générale de SGBD multimédia. Le traitement des données multimédia (texte, son, image, video, animation) diffère d'un type à un autre. Pourtant, des similitudes existent entre ces données du point de vue de leur gestion dans une base de données. Elles sont toutes de grandes tailles avec des structures complexes et une sémantique riche. Pour supporter des requêtes sur le contenu, la semantique des données multimédia doit etre extraite organisée, stockée et indexée.

Key Words: Multimedia database, SGML/HyTime document, Image database

1 Introduction

Multimedia is used for non-classic data such as formatted text, images, video, animation, and sound. The growing use of multimedia information systems has increased the need for efficient management of these types of data. The database research group of the University of Alberta is currently working on two multimedia projects. The first multimedia project [ÖIS+97], is an object-oriented multimedia database management system that can store and manage SGML/HyTime compliant multimedia documents. The system is capable of storing, within one database, different types of documents by accommodating multiple Document Type Definitions (DTDs). The second project, DISIMA (Distributed and Interoperable Image Management System) [OÖL+97], aims at building an image database system enabling content-based querying. Our current challenge is to merge the two prototypes for a full multimedia DBMS.

*This research is supported by a strategic grant from the Natural Science and Engineering Research Council (NSERC) of Canada.

A common aspect of all these data is the fact that they are big with complex structures and semantics. Processing is different from one type of multimedia data to another, and they need different input and output devices. The term *multimedia* is used for an application if it is able to handle at least one of these data. Applications combining several types of multimedia data are in fact a combination of several applications. However, similarities exist in the management of these various multimedia data. A common problem for all multimedia data is how to understand them? Raw multimedia data are a list of bytes that has to be preprocessed to identify the component objects (Figure 1). These objects can be someone's voice in a sound track, a person in an image or in a video, or a subject in a document. These objects can then be used for content-based queries. The objects are obtained by a manual and/or an automatic interpretation of the multimedia data. The objects have some temporal and spatial relationships that must be taken into account in the modeling, the querying and the presentation.

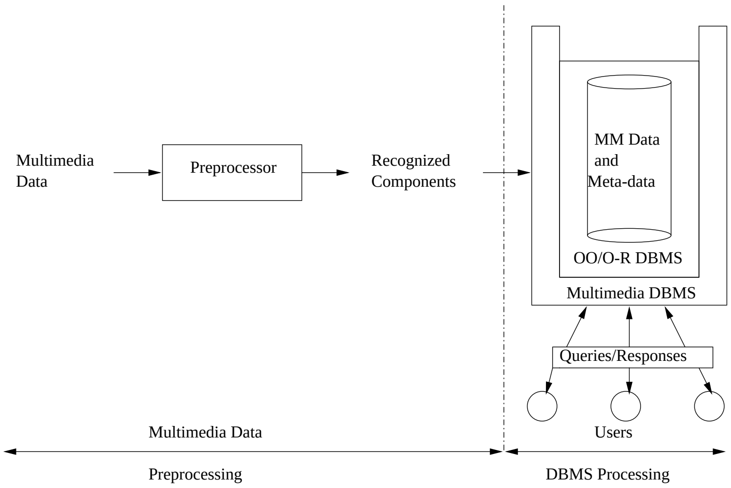

Figure 1: A Multimedia DBMS Framework

Some structured data often accompany the multimedia data. For an image, these structured data can be the photographer, the date, and the place. For a document, one would like to know the authors, the title, the publisher, etc. The raw multimedia data and all the related data have to be integrated and managed in the same database. The size and the complexity of multimedia data require special treatment in storage, querying, presentation, and transmission. Object or object-relational DBMS provide reliable support but a multimedia layer has to be added to fulfill all the requirements. Our prototypes are implemented on top of the commercial Object DBMS ObjectStore [LLOW91].

The solutions provided in the two projects are used to support the framework depicted in Figure 1. The remainder of this paper is organized as follows: Section 2 discusses the solutions used for the SGML/HyTime multimedia project, Section 3 presents the solutions for the DISIMA project, and Section 4 states our conclusion.

2 The SGML/HyTime DBMS Experience

The multimedia DBMS we developed is fully compliant with SGML, together with a subset of HyTime used primarily for the specification of hyperlinks and the synchronization requirements of continuous media components. The system, which is built on top of ObjectStore, consists of a kernel type system and a suite of document management tools. Text and images are stored in and managed directly by the DBMS; continuous media are stored on a continuous media file server (CMFS) and managed indirectly by the DBMS via meta-data that it maintains (Figure 2).

One set of document management tools provides the support for analyzing and incorporating new document type definitions (DTDs) into the system. This is accomplished by the DTD Parser that first validates a new DTD and then passes a parse tree to the Type Generator. The Type Generator uses the parse tree to automatically generate C++ code that extends the kernel type system with new types defining the new elements in the DTD. DTD elements can be specified in a manner that allows common element types to be consistently shared across multiple DTDs in the system. Compatibility of shared elements is automatically enforced. The new code is automatically compiled, built into libraries, and linked into applications that are in turn invoked to initialize selected databases with the new DTD and its associated schema.

The second tool set, for document entry, is concurrently built by the Type Generator during DTD incorporation. This process similarly involves C++ code generation to extend a kernel meta-type system. Singleton instances of the generated meta-types are instantiated in the target databases to enable the creation of object instances that represent element components of documents being entered into a database. Thus, code for a DTD-specific Instance Generator is generated by the Type Generator, incorporated into a DTD-specific library, and linked to an enhanced SGML parser to create a document entry application. This parser application is then able to process document instances by retrieving the appropriate DTD from the database. If the document is valid, it then passes a parse tree to the Instance Generator, which in turn retrieves the required meta-type instances from the database to create the new element objects from the parse tree data. Related tools provide for document removal or the removal of entire DTDs together with their associated documents. A simple document versioning system is currently being implemented.

2.1 The Type Systems

In an object-oriented DBMS, the design of the type system is comparable to schema design in a relational DBMS. Consequently, a very large proportion of the system design effort is directed toward type system design. The process of developing our SGML/HyTime DBMS can be divided into two distinct phases. In the first phase we designed and built a system for a single news-on-demand application which utilized a single Document Type Definition suitable for multimedia news. Under this approach we were able to defer dealing with many of the complexities of general SGML documents. Rather, we could focus instead on the underlying problems of representing different media types in the context of a single document structure. Moreover, this simplification of the design issues in the first phase facilitated co-ordination and integration work with our research partners in other laboratories across Canada who designed and implemented the QoS (Quality of Service) negotiation, synchronization, and continuous media file server components of the total system. In the second phase, we built upon our experience and generalized the database component to handle documents conforming to any SGML Document Type Definition.

2.1.1 Atomic Type System

During the design of Phase 1, we recognized the inherent distinction between types that could represent objects of different monomedia, irrespective of document structure, and types that could represent objects corresponding to the elements defined in the news-on-demand DTD. Capitalizing on this distinction, we developed a type system for representing the various monomedia that we considered essential for any multimedia application: text, images, audio, video, and synchronized text. These type definitions were totally independent of any SGML features, which came to be modeled by an element type system. They could be utilized equally well by any other multimedia document system as by an SGML-based system. Their defining characteristics were shaped largely by quality of service and storage considerations. The types representing non-continuous media, text and images, contained data members for storing the media data directly in the object itself. Such media objects were self-contained directly in the database. On the other hand, continuous media objects did not directly maintain the media data directly in the object. Instead, they contained meta-data and a universal object identifier which located the actual media data on a CMFS. The CMFS had the capability of delivering the data at the required rate in a synchronized manner with other media objects. Whether or not the monomedia data is stored directly in the database, the design of these types and their storage structures is critical for good performance.

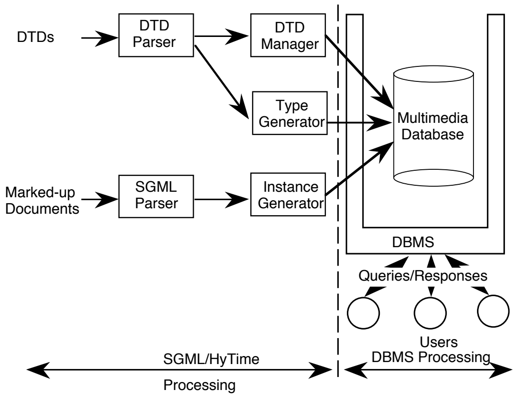

Figure 2: SGML/HyTime Processing environment

We call the type system for the monomedia types our atomic type system. It is designed as a rooted type system built upon the base supertype `Atomic`. Where possible, shared methods and data members are factored out into intermediate abstract types. This type system also utilizes some helper types, e.g., for representing QoS parameters, which are not rooted on `Atomic`. One very important and practical consequence of this modularization was the independence it provided our research partners in their development work. The work on the CMFS, QoS negotiation, and synchronization was largely confined to manipulating the atomic types. Co-ordination of our work on the element type system with modifications made by the other research groups on the atomic types was therefore facilitated by focusing our discussions on the interface between the two systems. Atomic type system details can be found in [Sch96, ÖIS+97].

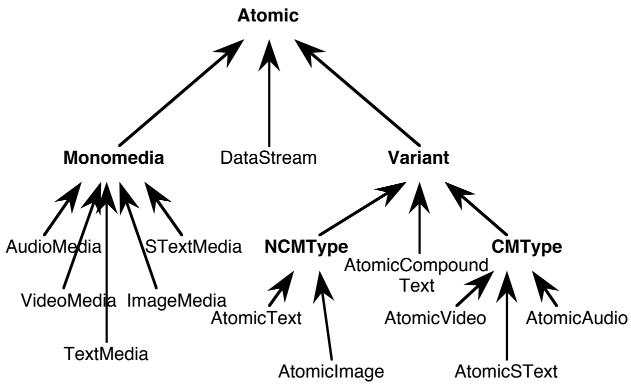

Figure 3: Atomic Type System

2.1.2 Element Type System

The element type system provides a uniform representation of the elements in the DTD and their hierarchical relationships. Each logical element in a DTD is represented by a concrete type in the element type system.

Design Decisions. During the design of the element type system in Phase 1, a number of fundamental design decisions were made which influenced the subsequent system developments. One such decision relates to the "shape" of the element type system. As a grammar, SGML is fairly flat, though it allows for a constrained composition of elements. This characteristic of SGML argues for defining a collection of element types unrelated by inheritance. However, such an approach does not take advantage of object-oriented modeling techniques, especially behavioral reuse. So rather than implementing a flat type system or a small set of highly generalized types to represent all elements, we settled on a highly structured type system rooted on the common base `Element` with higher level abstract types reused through inheritance and concrete leaf types to represent individual elements. This has the advantage of directly mapping the logical document structure to the type system in an effective way.

A second key decision concerned the storage system adopted for document text. For efficiency reasons, all the text of a document in Phase 1 was stored together as one text string in an `AtomicText` object [ÖEMV95]. Each textual component (e.g., paragraph, emphasis, etc.) had an annotation associated with it that indicated the start and end index of the object's text in the text string. Such annotations are not only useful for elements that contain textual data (#PCDATA in SGML) but also for other elements that have to be located within the text string to display them properly (e.g., figure, link). This text storage model avoids fragmentation of the document text into numerous objects and facilitates text retrieval and text searching.

While this model proved ideal for news-on-demand documents, its shortcomings become more apparent for DTDs having typically lengthy documents. Under this text storage model, the modification of text in a stored document requires modification of all annotations that follow or span (i.e., as a parent element) the changed text. Thus, a small change near the front of a long document requires annotation updates for nearly all the elements as well as (potentially) a string copy of the entire document text. Lengthy documents, such as books, would benefit by segmentation of the text into chapters or sections.

To address this problem in Phase 2, where any DTD can be incorporated into the system, the text storage model has been augmented by means of a composite atomic type, `AtomicCompoundText`, and a text segmentation facility that responds to the use of a "text-seg" attribute for selected elements in the DTD. This allows the DTD writer to specify which elements are to be maintained as separate text segments. The attribute can be given a default token value that can be over-ridden by the document author. Thus, while the DTD writer bears the initial responsibility for determining the desired granularity of text segmentation, the document author can over-ride the default specified in the DTD by specifying an alternative value for the "text-seg" attribute in the particular element's tag.

Generalization of the type system from a single news-on-demand DTD to multiple, arbitrarily complex DTDs entailed significant type system redesign as well as a disproportionate amount of work on the document processing side of the model. We extended the set of abstract classes in the Phase 1 element type system to form a kernel element type system from which an element from any DTD could be directly derived. These derived concrete element types were to be generated automatically in a DTD processing operation. Document entry processing was also much more complex. The document processing side of our DBMS model for the news-on-demand application simply consisted of a one-off parser that recognized conforming documents and appropriately called instantiation functions to populate a database. The current system requires a full SGML parser linked to generated code that is capable of instantiating element instances according to the parsed data.

The element type system can be further subdivided into three subsystems: the standard SGML types, HyTime types, and MM types (Figure 4). The standard SGML types are typically textual elements with annotations. HyTime types usually involve links and synchronization. MM types are used for monomedia components that can be represented in the atomic types. Details of these subsystems are in [Sch96, ÖIS+97].

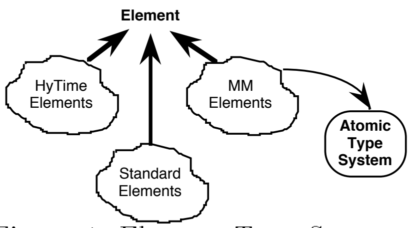

Figure 4: Element Type System

2.1.3 Meta-Types

In Phase 1 of the system there was no need for the implementation of meta-types for the purpose of instantiating document elements. It was enough for the news-on-demand document parser to directly "new" the appropriate element instances. This was hard-coded in the parser. With the arbitrary DTDs in Phase 2, an additional level of indirection was required. This indirection was achieved through meta-types.

The meta-type system is not directly a part of the element type system, but rather it shadows the element types with a roughly parallel type hierarchy, especially at the leaf nodes. For every concrete element type that is generated in the system, there is a corresponding meta-type. Like the built-in element types, which function as abstract supertypes for the concrete DTD-specific element types, a directed acyclic graph (a multiple inheritance hierarchy) of built-in meta-types provides the supertypes for the concrete meta-types that are generated by the Type Generator.

These meta-types perform two important tasks:

1. They store meta-information about the elements in the DTD, such as the element names, their attributes, and the supertypes from which the corresponding element types are derived. This information is necessary for instantiating the appropriate element instances and setting their attributes.

2. They define virtual create functions to instantiate persistent objects representing document element instances. These are referred to as instantiation methods. Single instances of these meta-types are created as persistent objects in a database destined for document instances conforming to a particular DTD.

2.2 DTD and Document Processing

As indicated, there was no need for any DTD processing with the single DTD in Phase 1. A simple initialization program could be run to initialize a database with the application schema and the object extents that were needed for news-on-demand documents. Moreover, the actual document processing needed to instantiate documents in the database was a simple one-off TCL parser and some instantiation procedures.

2.2.1 DTD Processing for Multiple Document Types

A DTD can be thought of as the definition of the syntax of a class of documents. In terms of the DBMS, it provides the schema for that class of documents.

In order to support a wide variety of applications with a multimedia DBMS for SGML documents, we concluded that it was essential to be able to create new types from any DTD introduced into the system. In other words, the system must be able to analyze new DTDs and automatically generate the types that correspond to the elements they define. Moreover, these types must become part of a database schema. In addition, the DTD must be an object in the database so that users can run queries like "Find all DTDs in which a 'paragraph' element is defined."

The components that have been implemented to support multiple DTDs are depicted in the upper left of Figure 2. A DTD Parser parses each DTD according to the SGML grammar defined for DTDs. While parsing the DTD, a data structure (parse tree) is built consisting of nodes representing each valid SGML element defined in the DTD. Each DTD element node contains information about the element, such as its name, attribute list and content model. If the DTD is valid, the parse tree is used by a Type Generator to automatically generate C++ code that defines a new ObjectStore type for each element in the DTD. Additionally, code is generated to define a meta-type for each new element type. Moreover, initialization code is generated and executed to instantiate extents for the new element objects and to create single instances of each meta-type in the specified database. A DTD object is also created in the database. This object contains the DTD name, a string representation of the DTD, and a list of the meta-type objects that can be used to create actual element instances when documents are inserted into the database.

One of the issues that had to be addressed with multiple DTDs is how to share element types with common features across DTDs. There is, however, no easy solution to this problem since it leads to the well-known semantic heterogeneity problem, studied extensively within the multidatabase community. Even when elements have the same names and structure, there is no way of ensuring that they are semantically equivalent.

Nevertheless, we have achieved a degree of type re-use in two ways. First, the design of the kernel type systems provides reusability. The atomic types, such as `AtomicImage, AtomicAudio, and AtomicVideo`, are used by any DTD that identifies those MM elements. More ubiquitously, `AtomicCompoundText` and `AtomicText` are used to store the textual component of all documents. In the element and meta-type systems, the kernel types encapsulate much code that is shared by the derived generated types. The second approach to reusability places the onus upon the DTD writer. We have provided a mechanism for the DTD writer to specify which elements in a DTD are to be treated as reusable across all the DTDs in the system. The writer identifies the semantic equivalence of elements across DTDs, provides them with the same names, and specifies a "reusable" attribute in the elements' definitions.

The system automatically manages such reusable element types. If the element named is not in the system it will be added, provided none of its child elements conflicts with an existing reusable element. All elements descendant from a reusable element's content model are implicitly marked as reusable. When a reusable type is added to the system, code is generated for it and its meta-type. This code is appended to a shared library of reusable types that gets linked into all the database applications that may need it.

After the Type Generator has produced the C++ code for the new DTD, the code is automatically compiled, built into shared libraries, and linked into applications for initializing databases and inserting documents into those databases. The DTD Manager then stores the DTD in the specified database as an enhanced DTD object that can be used for parsing documents and for other purposes. It also adds the new types to the database schema and creates meta-type instances for the DTD.

2.2.2 Document Processing for Multiple Document Types

Effective document processing is an area often overlooked in multimedia DBMS projects. Creating tools to validate and automatically insert documents is generally considered to be outside the scope of database work. To address this need, a significant portion of our efforts has been directed toward automatic document insertion.

The general architecture for this integrated component is depicted in the lower left of

Figure 2. The SGML Parser accepts an SGML document instance from an off-the-shelf authoring tool, validates it, and forms a parse tree. The Instance Generator traverses the parse tree and uses meta-type instances from the database to instantiate the appropriate element objects in the database. These persistent objects stored in the database can be accessed using the query interface.

3 The DISIMA Experience

The DISIMA research project deals specifically with the development of technology to facilitate the management of and access to images using a database management system. Specifically, we propose to develop an image DBMS which will provide the users with convenient services for uniform access to multiple, historically separated, and possibly heterogeneous image and spatial repositories.

3.1 The DISIMA Model

The DISIMA model addresses both image and spatial databases, allowing independence between image representations and applications and distinguishing the existence and identity of salient objects (logical salient objects) from their appearance in an Image (physical salient objects). The DISIMA model, is composed of two main blocks: the image block and the salient object block. We define a *block* as a group of semantically related entities.

3.2 The Image Block

The image block is made up of two layers: the *image* layer and the *image representation* layer. We distinguish an image from its representations to maintain an independence between them, referred to as *representation independence*. At the *image* layer, the user defines an image type classification similar to hierarchies in object type systems. This layer allows the user to define functional relationships between images. Figure 5 depicts a classification hierarchy for an application which manages news and medical images. These images can be classified according to specific criteria. The *NewsImage* class is specialized by three classes: one for images where a person has been identified (*PersonImage*), another for nature images (*EnvironmentalImage*) and the last one for the others (*MiscImage*).

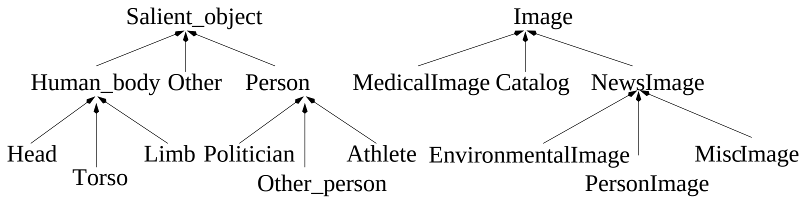

Figure 5: An Example of Image and Logical Salient Object Hierarchies

3.3 The Salient Object Block

DISIMA views the content of an image as a set of *salient objects* (i.e., interesting entities in the image) with certain spatial relationships to each other. The *salient object* block is designed to handle salient object organization. For a given application, salient objects are known and can be defined. The definition of salient objects can lead to a type lattice as in Figure 5. DISIMA distinguishes two kinds of salient objects: physical and logical salient objects. A *logical salient object* is an abstraction of a salient object that is relevant to some application. For example, an object may be created as instance of type *Politician* to represent President Clinton. The object "Clinton" is created and exists even if there is yet no image in the database in which President Clinton appears. This is called a *logical salient object*; it maintains the generic information that might be stored about this object of interest (e.g., name, position, spouse).

Particular instances of this object may appear in specific images. There is a set of information (data and relationships) linked to the fact that "Clinton appears in image i_1". The data can be his posture, his localization, or his shape in the image i_1. Examples of relationships are spatial relationships with regards to other salient objects belonging to image i_1. We create a *physical salient object*, (P_objectClinton_1) linked to the logical salient object Clinton, that refers to image i_1 and gives the additional information. If President Clinton is found in another image i_2, another physical salient object (P_objectClinton_2) will be created.

3.4 How to recognize the salient objects of an image

An image database can be seen as composed of two distict parts (Figure 6): the database component in charge of data organization and querying and the computer vision component that brings computational techniques to recognize objects. Despites progress in the computer vision field, automatic detection of objects is hard and application-dependent. As stated in [SKG98], the state of the art in computer vision does not permit automatic recognition of an arbitrary scene. Assume an object is detected by the image analysis soft-

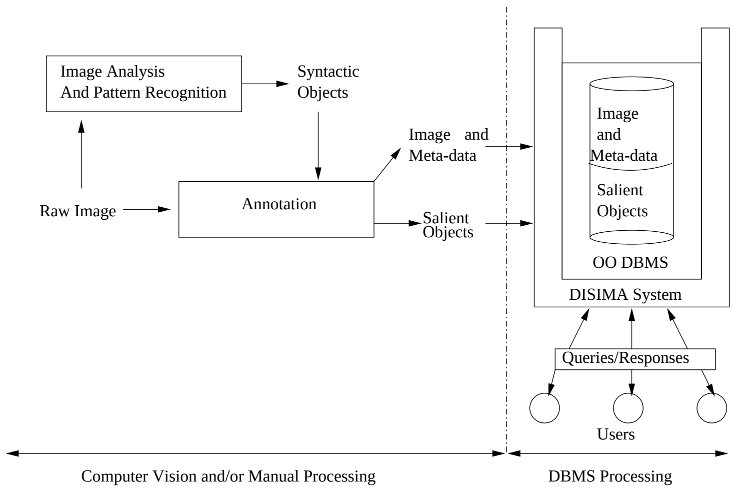

Figure 6: Image Processing Environment

ware. In the general case, this object is a syntactic object without any semantics. That is, it is a region of an image with properties such as color, shape and texture. Another challenge is to provide syntactic objects with semantics. Assume the object detected is a person. How can a computer assign a name to this person? This example explains why in most cases the image analysis is semi-automatic or manual.

One component of the DISIMA project is in charge of image processing and object detection. Our first concern was images with people. The image processing software detects the faces contained in the image with a minimum bounding rectangle (useful for spatial relationships) and a human-annotator assigns a logical salient object to the face. In addition, an image has some descriptive properties such, as date and photographer that have to be provided.

3.5 Querying images in DISIMA

DISIMA offers two ways to specify queries. A DISIMA query can be posed using the MOQL query language or the VisualMOQL interface. MOQL(Multimedia Object Query Language) [LÖSO97] is a general multimedia query language that extends OQL [CBB+97] by adding some spatial properties, temporal properties, and presentation properties for content-based image and video data retrieval as well as queries on structured documents. VisualMOQL is a visual programming language that we defined for the image component of MOQL ([OXÖ98]).

4 Conclusion

This paper has described a general framework for multimedia DBMSs. With reference to this framework we have described our experiences with developing two multimedia DBMSs: an SGML/HyTime DBMS and an image DBMS, DISIMA.

Our proposed research for the coming year will be in the application domain of electronic commerce. Our focus is to start development of multimedia smart catalog servers. This system will incorporate full multimedia with the inclusion of advertisement videos and audio "talk-overs". Since a monolithic system with all data stored at one location is not practical, we propose an open architecture that uses different specialized servers for different media types. A unifying generic interface layer on top of the specialized servers will provide for coherent user interactions while allowing underlying components to manage specific types of media objects and respond to the queries about them. This top layer will accept general user queries in MOQL and then decompose the queries so that they can be handled appropriately by the relevant underlying system. Initially we will integrate the SGML and DISIMA DBMSs that we already have. We have also begun to examine the integration of a video DBMS into DISIMA. This will be based on annotating key frames or composite frames extracted from a video by a tool such as Panorama [TAT97].

An associated project that we have begun [YOL97] will investigate the interoperability architecture for distributed heterogeneous repositories. This approach is based on the mediator/wrapper paradigm [Wie92] where information sources are wrapped so that their interfaces with the outside world are uniform and mediators are built to mediate the performance of various tasks across these repositories. The implementation will use a communication bus following both CORBA's Object Request Broker (ORB) [Sie96] approach and the COM/OLE [Bro95] approach.

References

[Bro95] K. Brockschmidt. *Inside OLE*. Microsoft Press, Redmond, WA, 1995. 2nd Edition.

[CBB+97] R. G. G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, San Francisco, CA, 1997.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The objectstore database system. *Communications of ACM*, 34(10):19—20, 1991.

[LÖSO97] J. Z. Li, M. T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proceedings of the 3rd International Workshop on Multimedia Information Systems*, pages 19—28, Como, Italy, September 1997.

[ÖEMV95] M. T. Özsu, G. El-Medani, and C. Vittal. An object-oriented multimedia database system for news-on-demand applications. *Multimedia Systems*, 3:182—203, 1995.

[ÖIS+97] M. T. Özsu, P. Iglinski, D. Szafron, S. El-Medani, and M. Junghanns. An object-oriented SGML/HyTime compliant multimedia database management system. In *Fifth ACM International Multimedia Conference (ACM Multimedia '97)*, pages 239—249, Seattle, WA, November 1997.

[OÖL+97] V. Oria, M. T. Özsu, X. Li, L. Liu, J. Li, Y. Niu, and P. J. Iglinski. Modeling images for content-based queries: The DISIMA approach. In *Proceedings of 2nd International Conference of Visual Information Systems*, pages 339—346, San Diego, California, December 1997.

[OXÖ98] V. Oria, B. Xu, and M. T. Özsu. VisualMOQL: A visual query language for image databases. In *Proceedings of 4th IFIP 2.6 Working Conference on Visual Database Systems - VDB 4*, pages 186—191, L'Aquila, Italy, May 1998.

[Sch96] M. Schöne. A generic type system for an object-oriented multimedia database system. Msc thesis, Department of Computing Science, University of Alberta. Available as Technical Report TR96-14, 1996.

[Sie96] J. Siegel, editor. *CORBA Fundamentals and Programming*. John Wiley, New York, NY, 1996.

[SKG98] A. W. M. Smeulders, M. L. Kersten, and T. Gevers. Crossing the divide between computer vision and data bases in search of image databases. In *Proceedings of 4th IFIP 2.6 Working Conference on Visual Database Systems - VDB 4*, pages 223—239, L'Aquila, Italy, May 1998.

[TAT97] Y. Tanigushi, A. Akutsu, and Y. Tonomura. PanoramaExcerpts: extracting and packing panoramas for video browsing. In *Fifth ACM International Multimedia Conference (ACM Multimedia '97)*, pages 427—436, Seattle, WA, November 1997.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. In *IEEE Computers*, pages 38—49, March 1992.

[YOL97] L.L. Yan, M.T. Özsu, and L. Liu. Accessing heterogeneous data through homogenization and integration mediators. In *Proceeding of Second IFCIS International Conference on Cooperative Information Systems (CoopIS'97)*, June 1997.