An Architecture for Distributed Multimedia Database Systems [1]

P.B. Berra C.Y.R. Chen A. Ghafoor C.C. Lin*
T.D.C. Little D. Shin*

Department of Electrical & Computer Engineering
Syracuse University
Syracuse, NY 13244-1240

*School of Computer and Information Science
Syracuse University
Syracuse, NY 13244-4100

Abstract– In the past few years considerable demand for user oriented multimedia information systems has developed. These systems must provide a rich set of functionality so that new, complex, and interesting applications can be addressed. This places considerable importance on the management of diverse data types including text, images, audio and video. These requirements generate the need for a new generation of distributed heterogeneous multimedia database systems. In this paper we identify a set of functional requirements for a multimedia server considering database management, object synchronization and integration, and multimedia query processing. A generalization of the requirements to a distributed system is presented, and some of our current research and developing activities are discussed.

[1] *Computer Communications*, (Special Issue: Multimedia Communications), Vol. 13, No. 4, May 1990, pp. 217-231. This research was supported in part by the NYNEX corporation through the New York State Center for Computer Application and Software Engineering (CASE) at Syracuse University.

1 Introduction

With the introduction of broadband communications and multimedia databases to business and consumers there comes a variety of new distributed applications. These include office information systems, geographical database systems, CAD/CAM systems, magazine production systems, medical information systems and a variety of military applications [CHR86]. Some systems have been implemented on an experimental basis [IRV88, KIM88, LIN89, NAF86, THO85, YAN88]. However, there are many limitations which must be overcome before these systems can be effectively utilized by the growing list of application developers.

One of the main objectives of multimedia information systems is to utilize current as well as future databases. A major challenge is to interconnect these databases and provide transparent services in the presence of considerable heterogeneity. Currently, there exist more than 3000 publicly available online databases, as well as a great number of private databases [ELS87]. Most of these databases can be accessed and potentially used for multimedia services. In a general multimedia environment, databases would contain various types of information such as text, voice, images, and full-motion video. In addition, they may be heterogeneous, with significant differences among local data managers, local data models and representation. Therefore, a primary objective in designing a distributed multimedia environment is to provide services that are transparent as to the nature and type of databases which need to be accessed to compose objects for the users. This can ensure that a user need not learn different database query and object manipulation languages.

As an illustration, suppose a doctor in an emergency room finds a patient in trauma with a severly damaged kidney. The patient needs an urgent kidney transplant. The doctor quickly retrieves a summary of the patient's medical history from his/her hometown 120 miles away. The history contains multimedia information such as X-ray's, CAT scans and other information. At the same time the doctor retrieves information regarding the availability of a kidney from a national database. The kidney which can be used must have certain physiological requirements which can be determined only by consulting the database containing the medical history of the patient. By simultaneously browsing through both databases, the availability of an appropriate kideney can be determined. In order to help the doctor make a decision, two different databases, presumably located at remote sites, must be accessed and the necessary information must be displayed in an integrated form to the end user. Furthermore, suitable database manipulation commands for browsing and updating the patient's medical history database must be available to the doctor. It should be noted that updating can be quite complex since storage of new X-ray's or CAT scans may need

specialized I/O devices for communicating information.

From the hypothetical example given above, it is clear that due to the diversified nature of users and information, a very important problem is how to represent, store, and fetch such information. An efficient solution to this problem may use the object-oriented approach in which each information unit is treated as an object with a topological structural description, or graph model, attached to it. Each information unit may be text, video, image, or data; or even a combination of them. Different kinds of information can be stored in different forms. For example, full-motion video signals as well as audio signals can be stored in an analog form on an optical disk. Subsequently, fetching objects and integrating them into a single object can require sophisticated manipulation. The object-oriented data representation problem has been extensively studied in the literature [KOS87]. However, the development of efficient distributed architectures to support such models has not been addressed.

The objective of this paper is to address a variety of fundamental design issues which are critical for the development of future distributed Heterogeneous Multimedia Database (HMD) system. Our remarks are partitioned into three main categories, namely, database management for multimedia information, object management in a distributed environment, and communication considerations to support broadband multimedia services. Accordingly, a set of functional requirements for an HMD system is derived, and a general design framework for the HMD system is presented. A methodology to manipulate a multimedia object in this system is also described.

The organization of this paper is as follows. In the next section the main functions of a multimedia server are specified. Based upon these specifications, a layered architecture of a centralized system is proposed in Section 3, which is currently being prototyped. This architecture provides a framework for the distributed system, which is discussed in Section 4. The concluding remarks are given in Section 5.

2 Functional Requirements for a Multimedia Server

A multimedia server is a source of diverse databases which can be used for a wide range of applications. Its main function is to provide an environment in which a user can develop suitable multimedia applications by using the information available in a network. Generally speaking, this environment should be transparent to the heterogeneous nature of databases being accessed by the users from the database server. In order to achieve these goals, a

number of functional requirements need to be specified for the server. The specifications for these functions in a centralized server, one without distributed multimedia sources, are the major theme of this section. Accordingly, a layered architecture of a centralized server is presented which is currently being implemented at Syracuse University. The distributed architecture based on this model is discussed in a later section.

The functionality of a multimedia server is much more complex than a conventional database manager. The multimedia server is not only responsible for managing complex information, which can be a form of text, images, audio and even full-motion video, but also must have the capability to integrate various information units to compose objects, which are of interest to the users. Object composition and manipulation can involve linking multiple objects, editing objects, enforcing consistency and protection for objects, and ensuring synchronization among information units which constitute objects. Due to the diverse nature of information, these functions are far more complex to implement than those provided by a conventional database system. The second important function of a server is to store and manage all kind of multimedia databases locally available on that server. These databases may be large archival databases available on optical disks or dynamic databases stored on high speed magnetic disks. We now describe their functional requirements in more detail.

2.1 Database Management Requirements

The major database mangement functions required by a server include file management and schema management for formatted data, scheduling of concurrent retrieval requests, implementing multidimensional browsing for video databases, synchronizing updates, handling of insertions, maintaining data integrity, version control and others. We now briefly describe each type of database and specify the supporting storage and management system.

2.1.1 Text and Formatted Data

Database systems based on relational or hierarchical model such as INGRES [HEL75] and SYSTEM R* [AST76] are well suited to managing formatted data. Database systems based on these models generally provide efficient access paths to data in secondary storage by maintaining some type of indexing scheme. Examples of general indexing schemes supported by these systems include variants of B-trees and Hashing. Queries based on a number of key values such as partial match queries are supported by using intersection operations. Tree

structure and hashing can be extended to N dimensional data in order to support partial match queries. Examples include k-d trees and multi-dimensional extensible hashing.

Each indexing scheme has its own advantages and disadvantages depending on the nature of query processing and data life time. When the database is static and most queries are based on few keys, an inverted index would be the best indexing scheme. On the other hand, when the database is to be frequently updated, and partial match queries based on a number of key values are common, surrogate files [BER87] and most dynamic multidimensional file structures can provide better performance in retrieval and update over other schemes.

Maintaining indexes for large unformatted textual data requires an even more complicated strategy as the size of the index itself can easily grow beyond manageable size when full text inversion is used for indexing. Although the access rate of this type data is not as crucial as the real-time deliverable data (video, audio etc.), the large index size could prevent the database system from performing real-time updating.

2.1.2 Audio and Music Data

Due to their nature audio waveform signals are usually sampled, encoded and then stored. In general, a higher quality encoding scheme requires a greater amount of memory space. An alternate approach to provide users with their desired audio signal is to use the approach of speech synthesis [WOE87]. However, this requires sophisticated processing and is usually slow. Compared to the speech signal, music signals are much more regular and structural. This implies that describing the abstraction of music signals at a high level is possible. A graph model has been proposed for this purpose [RUB87], where common musical notations are used to represent the structures of the graph.

2.1.3 Image Data

Many data structures have been proposed for storing images in a database system. These include pixel-oriented [CHO84], quadtrees [SAM84], R-trees [ROU85] or vector based [JUN85]. Most digitized images consist of a header representing format, size, and the number of bits used for representing a pixel, a set of color map values, and the pixel image data in order to allow browsing and conversion to another image format. FIVE (Format Independent Visual Exchange) is a collection of image manipulation packages developed by Bellcore to solve the incompatibility problem among various applications [BEL88]. FIVEtools includes utilities

for generating information about image files, displaying images, creating a new format from an old one (e.g. color to gray scale), compressing images, and editing images.

The image data is passive in nature, that is, it is generally accessed via unique id's without indexing for its contents, but requires large amounts of data space. To save storage space and enhance communication performance, the size of image data can be reduced by using data compression techniques.

2.1.4 Full-Motion Video Database System

Video data is unique in its nature in the sense that it can be totally in analog form containing both video frames and associated audio signals. The information can be stored in the same way as in video cassettes with the functions of replay, freezing frame, advancing, etc. However, in order to integrate this information with other data it must be digitized in order to prepare composite digital data packets carrying requested multimedia object information for the user. Therefore, analog to digital conversion and compression of video data needs to be carried out by the server. The equivalent inverse function needs to be performed at the user end. This signal processing can be avoided if the video information is prestored in a digitized compressed form. In digital form, data manipulation is much more flexible compared to analog form. However, irrespective of the nature of information, this service requires an enormous capacity for storage and very high access rate. Optical disks can be ideal for storing this information.

2.1.5 Physical Database Access Methods for Multimedia Data

As described above, the size of a multimedia data object can be very large, which may occupy multiple blocks in secondary storage. Furthermore, unlike conventional data models, an object can have a complex logical structure and can be shared among other objects at various levels. Therefore, the design of a physical storage structure for multimedia database systems is more demanding than that of conventional database systems. In order to improve the I/O performance, the use of parallel disk systems with suitable storage models has been investigated [COP85, KIM86, KHO88, THI87].

Current advances in parallel disk processing has accomplished dramatic improvements on data transfer rate from secondary storage. For example, the Connection Machine's parallel disk system (Data Vault) demonstrates data rate up to 40 megabytes/sec by incorporating

40 small disks [THI87]. The data transfer rate is actually proportional to the number of disks that can be accessed in parallel [KIM86]. Thus, by increasing the number of disks, it is technically feasible to achieve even higher transfer rates. Also optical disks using multiple beams have the potential for massive data rates [BER89]. However, there exists a five to six order of magnitude difference in access time between main memory technology and secondary storage technology. This cannot be alleviated by the use of parallelism and therefore is at the heart of the problem in data intensive applications.

One of our approaches to solving this problem is to store a large data object in physically adjacent blocks in secondary storage. Let s, r, and t be the seek time, rotational delay and data transfer rate of a disk drive, and let k and b be the total number of disks and the number of continuous blocks occupied by a complex object. Then, the data loading time can be represented by $T = s + r + b/(t \times k)$ which is considerably more efficient than that of randomly stored blocks ($T' = b \times (s + r + 1/(t \times k))$). The use of effective buffering also help increasing the performance by overlapping processing with I/O [BER79].

The physical data model also needs to support retrieval of relevant objects in response to user queries. Storage models for managing complex objects can be classified into three types [COP85, KHO88] including direct storage model (DSM), normalized storage model (NSM), and fully decomposed storage model (FDSM). DSM stores an entire object as a whole allowing fast retrieval of the object via object identifier, but does not support queries based on some key values and is not suitable for representing shared objects. Passive objects such as audio and image data can be stored using DSM in continuous blocks. FDSM and NSM are based on decomposition of complex objects into more simple structures. FDSM is a fully transposed storage model where each attribute is represented by a set of pairs of the form (object id, value), while NSM decomposes a complex object into a number of sets of tuples of the form (object id, value1, value2, ..., value k) by using a normalization mechanism. General index schemes can be applied to these models to support active data; for example, B-tree for FDSM and multidimensional dynamic files for NSM.

2.2 Object Synchronization and Integration Requirements

The main function of a server is to integrate information of units according to the semantic representation of a multimedia object, requested by a user. The relationship between various information entities may be temporal and thus time ordering needs to be maintained if browsing through databases is requested by the user.

We use a technique for the formal specification and modeling of multimedia composition with respect to inter-media timing called Composition Petri Nets (CPN) [LIT90]. The model is based on the logic of temporal intervals, and Timed Petri Nets. Through the model, the synchronization requirements of complex structures of temporally related objects can be easily specified.

2.2.1 Petri Nets

The presentation of multiple media is inherently a task which requires parallelism as well as sequencing. In this case the parallelism is in presentation and processing of multiple data streams. As mentioned, we have chosen a modified version of the Petri net [PET77], for a representation of the synchronization of multimedia entities.

For simple Petri nets, the time from enabling a transition to firing is unspecified and indeterminate. A class of enhanced Petri net models have been developed which assign a firing duration to each place [COO83]. These models are generally called *timed* Petri net (TPN) models, and map well to Markov performance analyses. Nonnegative execution times are assigned to each place in the net. With this scheme the notion of instantaneous firing of transitions is preserved, and the state of the system is always clearly represented during process execution (tokens are at all times in places, not transitions). This "augmented" model has the advantage of compactness of representation. The augmented model is supplemented with resource information associated with TPN models for the purpose of illustrating the use of the multiple media. This model is called the Composition Petri Net (CPN) [LIT90].

To illustrate multimedia synchronization using CPN, consider the following example. A multimedia slide presentation is to be represented. The presentation consists of a sequence of synchronized audio and visual elements of varying duration. Assume that there are n slides to present, and corresponding to each slide is a verbal annotation. On a time/resource line, we may represent the presentation as two streams of information which occur concurrently, as in Figure 1. Time is indicated on the horizontal axis. Resources are described by a dimension on the vertical axis indicating multiple threads of presentation which we call *space*. Using our Petri net representation, these activities are indicated by the Petri net in Figure 2.

For each place we have assigned the required presentation resource (device), and the time required to output the presentation data. The net indicates the points of synchronization between the two data streams. The transitions in the net indicate points of synchronization, and the places represent processing. For example, upon completion of the first verbal

Figure 1: Slide Presentation Timeline

Figure 2: Slide Presentation Petri Net

annotation, place t_1, represented by $talk_1$, unlocks its token. Because elements of the image (slide) and audio data streams have the same respective durations, the place s_1, indicated by $slide_1$, unlocks its token synchronously with place t_1. The common transition fires immediately, allowing the next image/audio pair to be presented.

It is possible to represent arbitrarily complex synchronization with this technique. Various degrees of granularity are possible as well. For example, full-motion video usually has an associated audio component. If one decomposes the video to individual frames, which are output at 30 per second [LIP80], matching audio may be synchronized in segments corresponding to frames in a Petri net structure as in the slide presentation example. Videodisc technology permits access of individual frames of full-motion video, which is important for building multimedia systems.

To provide a multimedia information service, the specification must be utilized in a manner to support the desired applicability. For an information storage and retrieval service, we must maintain the information necessary to synchronize any two data sources. This information clearly consists of a temporal relation, T_R, and the temporal intervals τ_α, τ_β, and τ_δ, between process entities [LIT90]. Given this information, we may completely construct the synchronized process interaction, modeled by a Petri net. Recalling the slide presentation

example, a process interaction is completely specified except for the temporal relations. For this example, between corresponding audio and image components the concurrent temporal relation *equals* is valid, with equal values for τ_α and τ_β. Between sequential image/audio pairs the sequential relation *meets* holds with time values τ_α and τ_β corresponding to the durations of successive pairs of elements.

We present a detailed example which elaborates the ideas and models previously presented. The example contains both static and dynamic data in the form of audio, image and textual data which require synchronization at the presentation level.

2.2.2 An Anatomy & Physiology Instructor: An Example for Data Integration

The Anatomy and Physiology Instructor is a simple multimedia application example based on the hypermedia paradigm and temporal relation specification. Figure 3 indicates the coarse grain hypermedia network for the Anatomy & Physiology Instructor. We assume that arcs in this figure represent links between informational units, represented by nodes, which contain lessons. Beginning at the **Start** node, users may browse via link selection or formulate explicit queries against the overall database.

When browsed or selected, information at a node is activated, causing its subsequent retrieval and presentation. For example, browsing **Alveolar Ventilation** assuming an initial starting point of **Respiration**, causes the information associated with that node to be presented. Fine grain synchronization is performed within the informational units.

A database structure is developed at two levels. At the coarse grain we desire the ability to select information units such as **Heart** or **Kidneys**. At the fine grain the synchronization constraints must be specified for concurrent presentation of various media elements. Each information unit may have different presentation requirements requiring distinct synchronization specification. Ideally, one specification suffices for all information units, simplifying the task of presentation design. For the general case, all nodes have different characteristics. We describe the synchronization requirements for one information unit; **Heart Muscle**.

To specify the synchronization requirements within an information unit a model for the presentation of information is required. For the example, the model used is based on the sample workstation screen indicated in Figure 4.

In this figure the informational unit **Heart Muscle** is presented. Note the different

Figure 3: Anatomy & Physiology Instructor Information Network [LIT90]

regions associated with the various media types including audio, video, text, image, and animated image (Views). Let us assume some characteristics and relationships between elements for this example. Figure 5 indicates possible relationships in time of the various elements of the example. This timeline representation shows that the textual component, the icon, and the "Location in body" images should persist for the duration of the informational unit presentation. The views are specified to change with time to provide a gradual rotation, or animation, of the organ selected, while the video and associated audio components begin presentation after a constant delay of τ_{13}. On the spatial axis are indicated the resources r_1 through r_8, associated with the workstation screen regions and the audio output.

Synchronization of the media elements is specified by the CPN model. By combining elements of the presentation in temporally related pairs, we create a CPN for the information unit shown in Figure 6. This CPN indicates the processing required for the presentation of a single Anatomy & Physiology Instructor information unit.

Between transition t_1 and t_9, we see a maximum of eight threads of presentation corresponding to the timeline of Figure 5. At the outset, seven threads are created as indicated by the initial transition, t_1. Places p_{16} through p_{19} represent static data which is active for the duration of

Figure 4: Workstation Instance [LIT90]

Figure 5: Anatomy & Physiology Instructor Timeline

Figure 6: Anatomy & Physiology Instructor CPN

the presentation. The two interacting threads, $p_1, p_3, p_5, p_7, p_9, p_{11}$, and $p_2, p_4, p_6, p_8, p_{10}, p_{12}$, represent the animation of two views of the organ, synchronized at points indicated by t_4, t_5, t_6, t_7, t_8, and t_9 of the CPN. The places p_{14} and p_{15} represent dynamic audio and video data, delayed by τ_{13}.

Having specified the synchronization requirements of an informational unit, we create a database schema to store the medical data and temporal relationships between data items. In assigning data to the database schema we must maintain temporal consistency. According to our CPN model, the pairings we have chosen, and the temporal relations used, we have a set of consistency requirements:

Duration of unit

$$
\begin{aligned}
&= \tau_1 + \tau_3 + \tau_5 + \tau_7 + \tau_9 + \tau_{11} \\
&= \tau_{13} + \tau_{14} = \tau_{16} = \tau_{17} = \tau_{18} = \tau_{19} \\
&\quad \tau_{14} = \tau_{15}, \tau_1 = \tau_2, \tau_3 = \tau_4, \tau_5 = \tau_6, \tau_7 = \tau_8, \tau_9 = \tau_{10}, \tau_{11} = \tau_{12}
\end{aligned}
$$

By storing data which satisfy these constraints, the system performs as specified by the CPN, without delays caused by inappropriate values.

Consider the interaction of a user with this multimedia system. If we use subnet replacement we may describe any node in the network graph as an equivalent Petri net place. Browsing or selecting nodes permits the sequential access of information units. By representing the selection activity as a query, a Petri net description of browsing a particular path is formulated. Figure 7 shows a net indicating the sequential access of the information units **Heart**, **Pulmonary Circulation**, and **Pulmonary Mechanics** in response to queries Q_1, Q_2, and Q_3. Using this notation one could assign bidirectional Petri net transition at each link (arc) in the information network to describe all possible browsing maneuvers. A complete Petri net description of the application results.

The examples presented clearly illustrate how we may use the CPN modeling strategy for many scenarios. Recapitulating, we have indicated the tools and strategy for the formal specification of multimedia object composition with respect to synchronization. Using CPN, multimedia objects may be choreographed in time and stored in a database indicated by a schema developed using the CPN. This database is utilized as a storage element supporting multiple access techniques including conventional query systems, hypermedia, functional, and procedure oriented methods.

Figure 7: Petri Net Representation of Browsing

2.3 Multimedia Query Processing

In this subsection, the query language and processing methodology for supporting a multimedia system are presented. Due to the diverse indexing techniques required for multimedia information, the conventional relational model has been recognized to be inadequate. In order to solve this problem, an object-oriented approach is used in this project, where each multimedia information unit is represented by an *object*, and an object may in turn consist of a number of subobjects (which can be text, image, video, audio, or a combination of them).

In object-oriented programming, the main functions are defined as *methods* which are procedures that implement operations defined on an object. For example, an image object may have methods to display, rotate, zoom in, and zoom out. A request that an object performs one of its operations is accomplished by sending a message to this object.

As mentioned earlier, data synchronization is one of the most important tasks in a multimedia system. It can be solved, at a low level, by defining a method called a *trigger*, in each object. A trigger executes automatically when its corresponding object is to be accessed. Therefore, the temporal relationship among objects (which, at a high level, are represented in Petri nets) can be established. At the programming level, a trigger can be implemented with a predicate and a body. When the predicate becomes true, the body is executed. *Inheritance* is a mechanism of sharing information among objects with similar behavior, which will allow a more efficient utilization of memory space in a multimedia system. Multiple and multiple level inheritance can also exist in order to support cross references.

Object *composition* is a way of combining related subobjects into a hierarchical structure, called a *complex object*, such that query operations can be performed directly on the complex object without going into the details of each subobject. In addition to simplifying query

processing, a complex object may also be a natural cluster in physical storage. This will reduce the required indexing time in accessing physical storage as well as memory space in supporting such indexing structures. Current multimedia systems only support user interface (such as [SUN87, THO85]) or simple object (in contrast to complex object) manipulation (such as [SCH85]). Research in the complex object query processing is still at a rudimentary stage.

In the following few paragraphs some multimedia query language syntax and semantics are illustrated as examples.

A class of objects, Personnel, can be declared using the command CREATE:

```
CREATE CLASS Personnel
        name    : { < last-name : Employee.last-name
                      first-name : Employee.first-name > }
        age     : {   Employee.age }
        salary  :     Employee.salary
        hobby   :     hobby.type
        skill   : { < type : skill.type
                      license : { < date : skill.date
                                    city : skill.city >
                                    <> modify } > }
        project : { < abstract : textindex
                      content : { <text : textindex
                                    image : imageindex >
                                    <> modify } > }
```

In this example, each person is represented by a complex tuple with attributes: name, age, salary, hobby, skill, and project, where each "<> modify" indicates that modifications on the corresponding field of information are prohibited. A graphical representation of the above example is shown in Figure 8.

In many cases, a user is only interested in a subset of information stored in the database. Therefore, it is necessary to introduce commands which will allow the creation of user views of the database. For example, the following command generates a user-view object schema (Figure 9) from the example described earlier:

```
    DECLARE VIEW introduction UPDATE
```

Figure 8: An Example of the Personnel Object Schema

Figure 9: User View Object Schema

```
FROM QUERY
        Retrieve ( name : X.name
                   Salary : X.salary
                   project : X.project.content )
        where X.name = "Smith Erbe"
```

where introduction is the reference point for this schema. Many other multimedia commands to allow users to open, delete, evaluate, navigate, move, modify, insert, etc., are also being developed.

3 Current Development of a Multimedia System

Given the requirement for multimedia system, we are developing a distributed system which contains three major nodes: an experimental platform, an eight-station SUN/4 cluster, and a Gould image processing machine. The networking of the three nodes is shown in Figure 10. Currently, a centralized multimedia system is being developed on the experimental platform node. After that, the generalization of such a centralized system to a distributed

Figure 10: Current Distributed System Configuration

one will be performed. The system developed on the experimental platform node will then be transferred to each workstation of the eight-station cluster. Moreover, all the computation-intensive image processing routines will be performed on the Gould image processing machine such that bottlenecks can be removed from the overall system. The detailed description of the experimental platform node is given below.

The architecture of the proposed centralized multimedia system is partitioned into three layers: database management system, data composition, and user interaction. Figure 11 shows a hierarchical ordering corresponding to the relationship among these components using the layered software specification approach. The interactive layer provides user access methods for object retrieval, while the data composition layer maps user level operations onto the data structures of the DBMS layer.

3.1 Multimedia Database Management Layer

The database management layer of our architecture consists of portions for managing both formatted (text and numeric) and unformatted data (audio, video, images). Note that we have decomposed the software architecture, allowing the data composition and database layers to be distributed throughout a multiple server system when we consider the distributed multimedia environment.

The DBMS component is responsible for all database management functions previously mentioned. Upon the request of a local controller, the appropriate databases are accessed and data is passed on. The architecture of this component consists of independent database managers for each database system which are directly interfaced to a multiprocessor system. An effective storage system can be a database machine such as the Teradata DBC 1012 [TER88] or IDM machine [BLC88] which can directly control the local databases. The controlling of structured text and data can be performed by conventional database managers while the non-formatted data can be placed under the management of specialized database controllers available on the machine. In either case, the database system needs to be interfaced with a local controller. This interface must be in the form of multiple I/O ports in order to support fast transfer of parallel data streams between the local controller and the high speed storage devices. An effective way to provide a low latency parallel interface is to use a multistage interconnection network between these systems. The network is a typical shuffle exchange network which can be build using off-the-shelf switching elements. Such networks can have fairly small latency times and have been successfully used for building large multiprocessor systems [DAV86]. This layer is of considerable interest to us since we are quite interested in the design and development of a multimedia database machine.

3.2 Object Composition Layer (Synchronization and Integration)

The data composition layer is responsible for enforcing the temporal synchronization and spatial integration requirements indicated by the multimedia database schema. Synchronization parameters, indicated by the CPN modeling tool described in Section 2, and stored in the database, choreograph the multimedia presentation. Similarly, spatial integration requirements are stored in the multimedia database using an object-oriented paradigm. This object composition component of the software system functions to bring together multiple types of data for the multimedia presentation. This layer is central to the the functionality of the multimedia system and interacts with both the DBMS and the interactive component of the

system.

3.3 Interactive Layer (Query Processing)

The interactive layer consists of the user interface including graphics and basic user operation functionality. In this layer we find database query operations such as relational manipulation and browsing via hypertext links. Also there are editing modules providing manipulation of various types of multimedia data including text, voice, graphics, and images.

The data of multimedia documents can be divided into two major parts: formatted data, such as integer and text, and unformatted data, such as image, graphic, and audio. In the existing multimedia systems, such as Intermedia [YAN88] and Telesophy [SCH85], the users can only specify conditions on the content and existence of formatted data. Access methods are only defined on these parts. Therefore, although we discuss a system for storing and retrieving the multimedia documents, query processing actually is only based on the formatted data of these multimedia documents. Further development of multimedia information management must include query on unformatted data as well as conditions on the existence and content of formatted data.

Query processing, which is a difficult problem in traditional DBMSs, becomes even more difficult when dealing with a multimedia document. Our query processing is based on a conceptual data model which is on top of the internal data model for the description of complex objects. This allows the system to exploit the rich structures of these objects during query processing. In addition, we combine different access methods for different data types found in the complex objects. The characteristics of the different access methods, plus the information about which access path to use for the particular components, allows the system to optimize query evaluation.

3.4 Hardware Configuration

At Syracuse University a prototype is being developed for a centralized Multimedia Server. Figure 12 shows the current hardware configuration of this server. The main component in this configuration is the SUN4/260 server, which has 500 Megabytes of disk space and a Parallax graphics 1280 series processor board [PAR88] with frame buffers for fast image processing. The Parallax board also allows real-time displaying and capturing of live video images. The system includes two additional SUN workstations. The SUN/386i workstation

Figure 11: Proposed Multimedia Application Platform

has 300 megabytes of disk space which is primarily used as a back up for the main server Sun4/260. The workstation SUN3/60 is a diskless unit and is used for developing multimedia applications. The capturing of live images through the Parallax is facilitated by a multimedia tool, known as PNeWS [PAR88]. A brief description of this tool is given in the next section. The color monitors attached to the Parallax board are used for displaying the graphics and video information. Additional interface units attached to the server include a QMS PostScript laser printer, a QMS color PostScript laser printer, a Sharp color scanner, a Sony video disk player, and a Sony video cassette recorder. The color scanner allows the creation of up to 300 dpi (dot per inch) images. The images created by the scanner can be used in both standard SunView and PNeWS environments. The video disk player and video cassette recorder are used as sources for live video. They can be accessed through PNeWS.

A number of multimedia software tools are currently resident on this configuration. These tools are expected to be used to implement the layered architecture of a centralized server shown in Figure 11. From the functionality point of view these tools can be classified into three categories; window management system, database management system, and image management system. We provide a brief description of these tools, which are currently loaded on the server.

3.4.1 Window Management

We are currently using NeWS (Network Extensible Window System) [SUN87] for providing high quality graphics and window management. NeWS is a distributed window management system based on the extension of the generalized concept of PostScript. The extension includes event management for user interactions, an interprocess communication handling facility among local and remote processes, and powerful graphic primitives for display. NeWS supports true Postscript-compatible graphics calls and handles such operations as zooming, scrolling, and panning. In a network or distributed environment, a workstation running NeWS is considered as a NeWS server which can be connected to client machines such as database machines or multiprocessor systems. PNeWS is an implementation of NeWS for Parallax Videographics Processors to provide video capabilities including display of live video, still video frames, and integration of graphics and video [PAR88].

Figure 12: Hardware Configuration

3.4.2 Database Management

We plan to use Ingres [HEL75] as the database management system for formatted text since it is the most widely available public domain relational database system in Unix. A relational database schema for handling multimedia objects is under development, which allows query support and object integration for multimedia objects. In our system development, the effect of changing the underlying database management system will be minimized so that various database management constructs can be exploited. We also plan to develop a high speed file access method based on surrogate files [BER87] for multimedia applications.

3.4.3 Image Management

For color image generation, we use the color scanner. This scanner uses a software tool called GBA Scanin [GBA89] in order to digitize images. This tool also allows enhancement of the quality of scanned images. The images created by the scanner can be used for the NeWS/PNeWS environment as well as the SunView environment. For image manipulation, our system uses Artisan [MED88], which allows creating and editing documents containing text, images, and drawn objects. Artisan supports various image formats including Artisan, TIFF, Targa, and SunRaster. Artisan also generates color PostScript files for high quality output.

3.5 An Application for the Multimedia Server

For the proposed centralized server, we are developing a multimedia application related to weather information. Currently, we are receiving visible and infrared satellite weather images. Visible images indicate cloud formations, whereas infrared images indicate cloud temperatures. The size of a satellite image is about 1.5 Mbytes ($= 1400 \times 1064 \times 8 bits$). Each image contains a header portion indicating time, date, sector number (one of approximately ten sectors), and an intensity bar to identify image contents. The system also receives radar precipitation images over direct telephone line from the local power corporation (Niagara Mohawk Power Corporation). A typical radar image is of the size 128 Kbytes ($= 512 \times 512 \times 4 bits$). In addition, various lightning images are also received, with a typical image also of size 128 Kbytes. The purpose of the use of these images and related information, is to develop various multimedia functions for weather applications. These include animation at various rates of sequences of related images, composition of various regional areas for

panning and zooming, and composing video clips of a local weather channel with the textual information.

4 Heterogeneous Distributed Multimedia Systems

For the distributed multimedia application domain it is expected that in the future data objects will be scattered across a network and will possess differing access requirements as dictated by commercial applications. Database servers will exist as components of these environments with varying performance characteristics and data formats, complicating the data integration process.

For a proposed heterogeneous distributed multimedia system we desire integration of data objects for ultimate presentation to the user, satisfying original presentation intent. Unlike the preceding centralized system, the choice for data integration must be determined from within the set of processors of the distributed multimedia system. For the centralized case, all composition and synchronization is performed within the capability (processing) of the workstation or centralized server. The alternate, distributed, scheme is to let a set of distributed database servers provide some degree of composition prior to communication of the multimedia objects [LIT89]. These two scenarios are indicated in Figure 13. Of course, the coordination of the composition process depends greatly upon the decomposition and distribution of the contents of the multimedia database. Suppose two objects are to be integrated into a single object for transmission to the user. If the two objects are stored by the identical database server then the selection of the appropriate composition node is obvious. However, if the data are stored on different servers, some mechanism must determine the appropriate optimal node to perform the composition.

Two other alternatives are apparent. First, each database server could maintain individual databases for each data type. Second, an intermediate composition processor/node could be assigned to coordinate the composition process for an object requiring data from a set of database servers. In the first case, we preserve performance of individual data type databases but maintain heterogeneity in the distribution of data which is good for reliability, security, and local access to data. In the second case, a single intermediate processor queries the necessary database servers for data required to build the desired object. Compared to the previous scenarios, an additional communication overhead is incurred with this scheme since no data is stored local to the intermediate processor. However, homogeneity in data of database servers is maintained, and no server composition bottleneck results as before.

Figure 13: Centralized vs. Distributed Data Composition [LIT89]

4.1 Heterogeneous Processing

Heterogeneity in data formats is a problem between different software environments as well as between machine types. The object-oriented paradigm offers the ability to hide machine dependencies and hardware characteristics from the system developer thus minimizing dependencies that affect the entire software system. Between machines with different concepts of data representations, canonical representations are used to minimize the number of translators required between data formats. XDR (eXternal Data Representation) [SUN88], is an example of such a canonical data scheme. If a canonical representation is utilized, it may be embedded in a class' methods, hidden from the application designer.

4.2 Communication Requirements for the HMD System

The integration of all multimedia services within a single broadband network poses new communication requirements for switching, multiplexing, transmission, control and processing of voice, audio and video signals. For example, full-motion video services using high definition television (HDTV) requires 1.2 Gbits/sec bandwidth. However, this rate can be reduced to 200-300 Mbits/sec using compression techniques [LIN86]. Services providing a broad range of still picture communication (images, graphs, maps, charts, etc.), depending on the picture resolution and color levels, can require transmission capability ranging from 50 kbits/sec to 48 Mbits/sec. The audio data rate can vary from 4 kbits/sec to 20 kbits/sec. Text data may be transmitted at considerably lower rates.

4.2.1 Full-Motion Video Communication Requirements

In order to support these diverse services, we use both circuit and packet switching. Since full-motion video services require real time communication with very low network delay, circuit switching is the most suitable mode, since it guarantees constant stability of transmission delay in communications and 100 percent availability of communications on given channels. The channel for such a service will be assigned between a user and the server, by the HMD controller. The overall bandwidth of the channel must be large enough to support the rest of the low data rate services (audio, text, images etc.) integrated with the full-motion video traffic. Since, there may be multiple sessions concurrently active in the network requiring such a service, it is essential that channel assignment be carried out intelligently in order to avoid inter-channel interference

4.2.2 Services with Low Broadband Communication Requirements

Services requiring still images, graphics and maps etc., require varying transmission rates depending upon the resolution of the picture and the number of colors. For example, a 320×200 low resolution picture with 256 colors requires about 0.5 Mbits/picture. The transmission rate of 50 kbits/sec will transfer 0.1 picture/sec. On the other hand, a transmission rate of 48 Mbits/sec can deliver 2 pictures/sec each having a 1000×1000 resolution. Since there is no motion present, network delay may not be a serious consideration. For these services, high speed packet switching techniques can be employed, which can also be employed for audio traffic. However, care must be taken on the delivery of voice packets since this type of traffic also has some network delay requirements for real-time delivery. It is worth noting that in order to provide a reliable delivery mechanism, virtual circuit communication is essential for the services mentioned above which use packet switching.

4.2.3 Services with Narrowband Communication Requirements

For services requiring narrowband communication such as structured text and formatted data, packet switching techniques can be used. Similarly, all the control as well as protocol signals between users, the controller and servers can be transmitted using packet switching. Virtual circuit communication can be employed to provide reliable and error free transmission. Most of the control packets can be transmitted as datagrams, using a special control channel designated for this purpose in the network.

4.3 Proposed HMD Architecture

The global architecture for the proposed HMD system is shown in Figure 14. Generally, the HMD system consists of n servers (i.e., $S1$, $S2$, ..., Sn), m users (i.e., $U1$, $U2$, ...Um), a communication network, and a central controller. Both n and m are dynamic and can be changed easily according to the actual needs in the application environments and $m >> n$. To simplify the discussion, for the time being, a symmetrical architecture is assumed, that is, all the servers are the same in their hardware and architectural design. Besides, heterogeneous objects can be represented in the system in order to significantly increase the accessibility. In other words, if a desired server is busy, an incoming query can go to another server to access the same desired object. Although the major concern becomes maintaining the load balancing among servers, difficulties occur in maintaining object consistency and

Figure 14: A Distributed Multimedia Database System

allowing a low-level object to be shared by several high-level objects. We now describe architecture of various components of the HMD system.

4.3.1 User Workstation

The main function of a workstation is to display multiple data to the user in the desired format. For example, if the video data is in digital form, it needs to be converted into analog form for display. Similarly the image data can be in many possible forms. Accordingly, processing needs to be carried out before the data can be displayed. It is therefore, essential that this component be a smart terminal with significant processing and storing capabilities. Furthermore, it should be able to interface with the communication network and support communication protocols in order to interact with the central controller and multimedia servers.

4.3.2 Communication Network

The main function of the network is to interconnect geographically dispersed servers and users and provide communication support for the services we mentioned above. Therefore, the use of optical Metropolitan Area Network (MAN) technology is the most feasible solution for such a high speed data communication requirement. Since a server in the HMD system can be a multiprocessor system, multiple ports can be made available for networking or a cluster of databases interconnected through a Local Area Network (LAN). A MAN can effectively serve as the backbone network interconnecting all these sites. A number of organizations such as the CCITT, the IEEE and the ISO are in the process of defining standards for networks, protocols and interfaces in order to support Broadband Integrated Services (BIS). The widely supported broadband integrated services digital network (B-ISDN) for MAN technology is the dual bus Queued Packet and Synchronous Switch (QPSX) [NEW88]. QPSX MAN has many features such as high reliability and support for arbitrary network size and speed. Therefore, such a network can be employed effectively in the proposed HMD system. In order to provide real-time interactive multimedia services, B-ISDN should have a low delay and a high data transfer rate on the order of Gbits/sec. With the current state-of-the-art in optical fiber technology, networks with capacities approaching 2-5 Gbits/sec using wavelength-division multiplexing have been reported. We, therefore, envision that the network used for the HMD system will be capable of providing such a high data rate.

An efficient use of the network and common channels is critically dependent upon the Media Access Protocol (MAP) and the interface mechanism to the network. The interface between a server and the network must permit reasonably high data rates to avoid congestion of data. Furthermore, it is desirable that standard interfaces which operate in the asynchronous mode be used so that the software changes to the server's operating system are minimal. A number of standards for MAP have emerged recently for high speed optical networks such as IEEE 802.3 (CSMA/CD), IEEE 802.4 for broadband services [IEE85] and CCITT's I-400 series [GIF86]. Any one of these standards can very well be adopted for the proposed HMD system.

4.3.3 Controller for Distributed Objects

The distributed functions such as object management, server allocation, etc. can be carried out by a central controller as shown in Figure 14. A user's request carrying the necessary

parameters for the services to be provided, is first received by the controller. The controller interprets the query in order to identify servers which have the desired objects and then establishes a binding between the user and the associated servers. The controller determines in which server a specific object is stored (or servers in case the object is distributed and stored over multiple servers). It then selects the most suitable server which we will refer to as the *master server* in case a multi-server object is requested. The rest of the participating servers will be referred to as *secondary servers*. Such an identification is directly related to the type and nature of databases available at various sites. The decision for selecting the master server is based on the current load on the servers and the cost of data movement among servers. The major factor for the decision is the latter one since it severely affects the overall query response time as well as the amount of data traffic in the network. It then determines the procedure for the operations on an object which requires multi-server access.

Additional functions carried out by the controller include prioritizing queries, detecting possible temporary constraints in performing object integration at the global level, interpretation of the function of each query at a high level and finally, guiding the object communication of all the participating servers. The controller also needs to support communication protocols in order to have reliable communication between itself and the users as well as with the servers. Discussion of the communication protocols is given in a subsequent section. The controller is also responsible for all the network management functions such as server initialization, authentication, logging the generation of new databases and multimedia objects, assignment of network channels for supporting concurrent sessions, etc. The architecture of the controller can be a uniprocessor having sufficient memory.

4.3.4 Multimedia Distributed Server

The node of a multimedia server for a distributed environment is essentially the same as the centralized server with some additional functional requirements. For example, once a server is selected as a master server for a user, it becomes responsible for the integration of the databases (both local and external). Secondary servers are notified by the central controller to allow access to their databases, which serve as external databases to the master server. Upon this notification, the master server establishes sessions with the secondary servers in order to maintain an interactive environment and to fulfill the user's request regarding manipulation of the external databases. The master server upon establishing the connection starts receiving external data and integrates it with its own local data according to the semantic representation of the multimedia object. The relationship between various

data entities may be temporal and thus time ordering needs to be maintained if browsing through databases is requested by the user. It should be noted that the server must have large memory in order to hold external data while integrating it with the local data. Also, large memory will allow browsing through external data without incurring retransmission of that data from the secondary servers.

Another additional requirement for a server in a distributed system is its ability of networking, that is, it needs to support a peer-to-peer communication mechanism. This mechanism can be supported by a hierarchy of communication protocols, for which there exist many commercial and international standards. These include ISO reference model [TAN81], CCITT's X- series, and SNA by IBM etc. These protocols provide both virtual circuit as well as datagram services needed for the proposed HMD system. These can also support the application services needed between the user and the server. The main function of the protocol hierarchy is to provide a reliable error-free communication mechanism between two remotely located entities. The interaction between a user is mainly on the basis of a virtual circuit while the transmission of integrated data needs circuit switching, as described in a previous section. The interaction between the controller and the other servers is also supported through the use of these protocols.

To obtain proper synchronization (speed matching) between the network and the server and to avoid congestion of data at the network interface point, the server needs to have multiple network ports. For a smooth flow of data between the server and the network ports, efficient interface protocols also need to be implemented by the server.

In order to perform proper load balancing and make effective use of local resources the architecture of the server is assumed to be partitionable into sets of clusters (see Figure 15). The basic philosophy of the partitioning of the architecture into clusters is that a partitioned cluster can be assigned to those users who require more or less the same type of external and local data. The major advantage of this scheme is that various external data stored temporarily in the local memory of a cluster can be effectively integrated to serve multiple users, without incurring repeated transmission of the same data from secondary servers. Clearly, the performance can be greatly enhanced through the proposed scheme. The resulting architecture to support these functions is hybrid in nature with the following properties:

- The controller consists of multiple processors and has a large global memory which is partitionable and shared among all the processors.

- A high speed interconnection network is present between the processors and the main memory modules.
- There is another high speed interconnection network between the database machines and the main memory modules. This network is also used to provide multiple ports to the optical network.
- Memory management, partitioning of the system, assignment of resources, and the network interface is under the control of the server's operating system.

4.4 Multimedia Access and Retrieval

4.4.1 Object Graph

To perform global operations, we need to define the internal data representations and their memory storage policy. Two-level object graphs are maintained in the system, i.e., local and central, where a local object graph is maintained by the local controller of each server and a central object graph is maintained in the central controller. Consider the example shown in Figure 16, where, for the reason of simplicity, a two-server system is assumed and each server is assumed to have four memory modules, i.e., $MM1_i$ and $MM2_i$ $(i = 1 \ldots 4)$. In order to clearly present our global architecture, we will describe each information token at the abstract object level. As an example, object $O8$ consists of objects $O1$ and $O2$ (or partial $O1$ and $O2$), $O1$ is stored in $MM1_1$ and $O2$ is stored in $MM1_2$ of Server 1. Similarly, $O15$ is stored in $MM2_1$ and $MM2_4$ of Server 2. Note that $O22$ and $O23$ are stored partially in Server 1 and partially in Server 2. The term multiple server object is used for them.

First, the term local graph for server i is used to represent a subgraph of the object graph, where each object in this subgraph is stored in Server i. Examples of local graphs are shown in Figure 16 by dashed boxes. Note that if we store the entire object graph in the central controller and perform the search of each object in it, the speed of the central controller will be very slow and could easily become the bottleneck of the whole system. Thus, the entire object graph is partitioned into many local object graphs and one central graph. And, at the same time, the required manipulations on the object graphs are then distributed into local servers. Therefore, it is clear that we need to have a local controller for each server to be in charge of the operations on the corresponding local graph.

A central object graph consists of all the multiple server objects and non-descendent

Figure 15: Architecture of a Server

Figure 16: Object Graph

Figure 17: Central Object Graph

objects in all local object graphs and the corresponding server(s) for each object. For example, the central object graph for the example in Figure 16 is shown in Figure 17, where descendent objects such $O1, O2, \ldots, O7$ are not included in the central object graph and non-descendent objects in local object graphs such as $O19$, $O20$, and $O21$ are included. Note that although $O19$ is a descendent of $O23$ (which is a multiple server object), it is still a non-descendent object in the local graph of Server 2. Each object in the central object graph is at the highest-level in the database system. For example, a system may classify all the information into education, medical, entertainment, transportation, etc., and each of them will an object in the central object graph. It is clear that, in the central object graph, only single level objects are used. Thus, only a simple linear time search is required.

4.4.2 Global Object Manipulation

With local and global object graphs, a user's query can be manipulated in the following way. It is reasonable to require a user to understand in which category (i.e., in which object of the central object graph) a desired object has been classified. If we use the example mentioned above, a user needs to know whether the desired object is related to education, medical, entertainment, etc. Therefore, it is realistic to assume that a user can access the central object graph from the I/O device. Thus, to access an object O_l in the database, the user needs to specify:$(O_c$: $O_l)$, where O_c is the ancestor of O_l and is in the central object graph. A typical user query is as follows: $(f\ (O_{c1} : O_{l1})\ (O_{c2} : O_{l2})\ \ldots)$ where f is the function of this query. The user simply places the desired object name in the network, the central controller searchs for O_c in the central object graph and then forwards the query to the corresponding server. For a multiple server query, a distributed object integration policy is required. That is, the central controller would select one of the related servers to be the master server

and others secondary servers, such that objects from secondary servers are transferred to the master server through the global network and integrated into a global object requested by user. The criteria for the central controller to select a master server has already been addressed. It is clear that if required object is a video signal, then the server containing this object is very likely to be selected as the master server. Note that, because all the servers are remote to the central controller, it will not be efficient to directly perform object integration in the central controller. When new objects are to be stored in the database, existing objects are to be modified, or the structural connections of some existing objects are to be changed, the local object graphs and possibly also the global object graphs need to be updated. Note that to guarantee object consistency, each object in the central object graph requires the functions of "lock" and "unlock". For example, consider the case where an existing object (O_c: O_l) is to be updated. First, the query containing this object will be processed by the central controller which will first check the function of this query, forward this query to servers, and then immediately lock O_c. As soon as the update operation is finished, the local controller sends a control signal to the central controller to unlock O_c and, at the same time, sends an acknowledgement signal to the user. Of course, it is also possible to distribute the functions of "lock" and "unlock" to local controllers. However, it would complicate the process of a multiple server object.

4.4.3 Local Object Manipulation

Once a user's query has been transferred to a server, search is performed to locate the object in the local object graph; then decomposition (toward the direction of leaves in the local object graph) starts by traversing the local object graph until reaching the DBMSs. For example, suppose a query needs to access $O11$ in Figure 16. Search for this object is performed on the local object graph, then $O11$ is decomposed into ($O5$, $O6$, $O7$) which is then decomposed into (($MM1_2$, $MM1_3$), $MM1_1$, $MM1_4$). Next, the subobjects from $MM1_1$, $MM1_2$, $MM1_3$ and $MM1_4$ are fetched into the local controller and integrated into a single user object. Inside a server, because all the DBMSs are local to the local controller, object integration can simply be performed in the local controller. In other words, the approach of using master and secondary server is no longer necessary here.

Because there will be many queries occurring simultaneously in a server and some of them may have predefined temporal ordering, it is necessary to attach a time variable to each object in a query to guarantee that when a user receives the response the order is correct. This will require intelligent object integration inside a local controller. Note that,

in order to reduce the load of the central controller, most of the temporary synchronization operations are performed in the local controllers. A data flow approach can be used for object integration. Each object at any abstraction level is represented by a template. Once all the required parameters are available, the output object (integrated object) is forwarded to the template of its father node. This operation continues until it becomes the object which the user requested. Therefore, it is very important to set up a data flow graph for each object. Since the recursion problem occurring in traditional computing problems does not occur in this case, the analysis will not be as complex as in a data flow computer.

5 Conclusions

In this paper, we have presented a set of functional requirements for a multimedia server. These requirements have been partitioned into multimedia database management, object synchronization and integration, and multimedia query processing for object manipulation. Accordingly, we proposed a three layer architecture for a centralized server which is currently being prototyped. At the database level there are a myriad of interesting research problems that must be addressed. Foremost among these are storage, access, updating and integrity of various types of image and video data. Data synchronization and integration is a critical problem for object composition for which we have developed a Petri net model. This model is part of the overall multilevel object oriented approach taken to ensure uniform continuity from the user level down to the database system level. The centralized model provides a framework for the consideration of a distributed multimedia architecture which we also discuss in the paper. We conclude that the issues involved in the development of distributed multimedia information systems are more challenging than those of the centralized model and extensive research will be required.

References

[AST76] M. M. Astrahan et al., "System R: Relational Approach to Database Management", *ACM Trans. Database Systems*, vol. 1, no. 6, June 1976.

[BAN87] J.H. Banerjee et al., "Data Model Issues for Object-oriented Applications," *Trans. Office Information System,* pp. 3 - 26, January 1987.

[BEL88] "FIVE User's Manual," BELLCORE, Morristown, NJ., 1988

[BER79] P. B. Berra and E. Oliver, "The Role of Associative Array Processors in Database Machine Architecture," *IEEE Computer*, vol. 12, no. 3, pp. 53-61, March 1979.

[BER87] P.B. Berra, S.M. Chung, and N.I. Hachem, "Computer Architecture for a Surrogate File to a Very Large Data Knowledge Base", *IEEE Computer*, vol. 20, no. 3, pp. 25 - 32, March 1987.

[BER89] P.B. Berra et al., "The Impact of Optics on Data and Knowledge Base Systems," *IEEE Trans. Knowledge and Data Engineering*, vol. 1, no. 1, pp. 111-132, March 1989.

[BLC88] "The Intelligent Database Machine", Product Literature, Britton Lee Corp., Los Gatos, CA, 1988.

[BOW88] N.Bowen, C. Nikolaou and A. Ghafoor, "Hierarchical WorkLoad Allocation for Distributed Systems," *Proc. 17-th Int. Conf. on Parallel Processing System*, St. Charles, Illinois, August 1988.

[CHO84] M. Chock, A.F. Cardenes, and A. Klinger, "Database Structure and Manipulation Capabilities of the Picture Database Mangement System (PICDMS)," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 4, pp. 484-492, July 1984.

[CHR86] S. Christodoulakis and C. Faloutsos, "Design and Performance Considerations for an Optical Disk-Based, Multimedia Object Server," *IEEE Computer*, vol. 19, no. 12, pp. 45-56, December 1986.

[COO83] J. E. Coolahan Jr. and N. Roussopoulos, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," *IEEE Trans. Software Engineering*, vol. SE-9, no. 5, pp. 603 - 616, September 1983.

[COP85] G. P. Copeland and S. N. Khoshafian, "A Decomposition Storage Model," *Proc. SIGMOD '85*, pp. 268-279, 1985.

[DAV86] E. Davidson, D. Kuck, D. Lawrie, and A. Sameh, "Supercomputing Tradeoffs and the CEDAR System," CSRD Rep. No. 577, *University of Illinois*, Urbana, May 1986.

[ELS87] "Directory of Online Databases 8(1)," Caudra/Elsevier, New York, January 1987.

[GBA89] "GBA-Scanin: Installation and User Guide," GBA, 1989.

[GHA88] A. Ghafoor and F. Y. Farhat, "Dynamic Concurrency Control Algorithms for Large Distributed Database Systems," To appear in *The Computer Journal*, Cambridge University Press.

[GIF86] W.S. Gifford, "ISDN User-Network Interfaces," *IEEE J. Selected Areas of Communications*, vol. SAC-4, pp. 343 - 348, July 1986.

[GUY81] A. C. Guyton, "Text Book for Medical Physiology," W. B. Saunders Co., Philadelphia, 1981.

[HEL75] G. Held, M. Stonebraker and E. Wong, "INGRES - A Relational Database System", *Proc. 1975 NCC*, Anaheim, Ca., June 1975.

[IEE85] ANSI-IEEE Standard ISO Draft International Standard, Std 802.3-1985, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, 1985.

[IRV88] J.H. Irven et al., "Multi-Media Information Services: A Laboratory Study," *IEEE Communication Magazine*, vol. 26, no. 6, pp. 27 - 44, June 1988.

[JUN85] E. Jungert et al., "Vega - A Geographical Information System," *Proc. Scandinavian Res. Conf. Geographical Information Systems*, Linkoping, Sweden, June 1985.

[KHO88] S. Khoshafian, P. Valduriez, and G. Copeland, "Parallel Query Processing for Complex Objects," *Proc. 4th Data Engineering Conference*, pp. 202 - 209, February 1988.

[KIM86] M. Y. Kim, "Synchronized Disk Interleaving," *IEEE Trans. Computers*, vol. C-35, no. 11, pp. 978-988, November 1986.

[KIM88] J. Kimsey, "Museum Cataloging," *Computer Graphics World*, May 1988.

[KOS87] K. Kosaka et al. "An Experimental Mixed-Object Database System", *Proc. IEEE Computer Society Office Automation Symposium*, Gailthersburg, MD, pp. 57 - 66, April 1987.

[LIN86] L. Linnell, "A Wideband Local Access System Using Emerging Technology Components," *IEEE J. Selected Areas of Communications*, vol. SAC-4, July 1986.

[LIN89] M. A. Linton, J. M. Vlissides, and P. R. Calder, "Composing User Interfaces with InterViews," *IEEE Computer*, vol. 22, no. 2, pp. 8 - 22, February 1989.

[LIP80] A. Lippman, "Movie-Maps: An Application of the Optical Videodisc to Computer Graphics," *Computer Graphics*, pp. 32-42, July 1980.

[LIT89] T. D. C. Little and A. Ghafoor, "Multimedia Object Composition in a Distributed Environment," to be submitted to IEEE Trans. Parallel and Distributed Systems, 1989.

[LIT90] T. D. C. Little and A. Ghafoor, "Multimedia Object Models for Synchronization and Databases," to be published in Proc. 6th Data Engineering Conference, Los Angeles, CA, February 1990.

[MED88] "Artisan User's Guide," Media Logic Inc., 1988.

[NAF86] N. Naffah and A. Karmouch, "Agora - An Experiment in Multimedia Message Systems," *IEEE Computer*, vol. 19, no. 5, pp. 56-67, May 1986.

[NEW88] R.M. Newman, Z. L. Budrikis and J.L. Hullett, "The QPSX Man," *IEEE Communication Magazine*, vol. 26, no. 4, pp. 20-28, April 1988.

[PAR88] "Parallax Graphics 1280 Series Processor Manual," Parallax Graphics, September 1988.

[PET77] J. L. Peterson, "Petri Nets," *Computing Surveys*, vol. 9, no. 3, pp. 225-252, September 1977.

[RUB87] W. B. Ruberstein, "A Database Design for Musical Information," *Proc. '87 SIGMOD*, December 1987.

[ROU85] N. Roussopoulos and D. Leifker, "Direct Spatial Search on Pictorial Database Using Packed R-trees," *Proc. '85 SIGMOD*, pp. 17-31, May 1985.

[SAM84] H. Samet, "The Quadtree and Related Data Structures," *Computing Surveys*, vol. 16, no. 2, pp. 187-260, June 1984.

[SCH85] B.R. Schatz, "Telesophy," Bell Communications Research TM-ARH-002487, August 1985.

[SUN87] "NeWS Manual," Sun Micro Systems Inc. Part No. 800-1632-10, March 1987.

[SUN88] "Network Programming," Sun Micro Systems Inc. Part No. 800-1779-10, February 1988.

[TAN81] A.S. Tanenbaum, *Computer Networks*, Prentice Hall, 1981.

[TER88] "DBC 1012," Product Literature, Teradata Corp., Los Angeles, CA., 1988.

[THI87] "Connection Machines Model CM-2 Technical Summary," Thinking Machines Co. Technical Report HA87-4, April 1987.

[THO85] R.H. Thomas et al., "Diamond: A Multimedia Message System Built on a Distributed Architecture," *IEEE Computer*, vol. 18, no. 12, pp. 65- 78, December 1985.

[WOE87] D. Woelk, W. Luther, and W. Kim, "Multimedia Approach and Database Requirements," *Proc. Int. Conf. on Office Automation*, 1987.

[YAN88] N. Yankelovich, et al., "Intermedia: The Concept and the Construction of a Seamless Information Environment," *IEEE Computer*, vol. 21, no. 1, pp. 81-96, January 1988.