분산 멀티미디어 시스템을 위한 범용 멀티미디어 처리 모델의 객체지향, 클라이언트-서버 구조

김 두 현[†] 임 영 환[††]

요 약

본고는 분산 멀티미디어 시스템 환경에서 여러기지 응용 서비스를 지원하기 위한 범용 멀티미디어 데이타 처리 모델을 제시하고 그것을 구현하기 위한 객체지향, 클라이언트 서버 구조에 대하여 기술하였다. 이 모델은 분산 멀티미디어 시스템에서 필요한 원격 데이타의 이용을 위한 통신망 투명성과 실시간 멀티미디어 입출력, 그리고 미디어의 통합이나 동기화 등의 멀티미디어 처리를 지원하는 범용 모델이다. 이 모델은, 스트림 계층, 멀티미디어 프리젠테이션 계층, 하이퍼 프리젠테이션 계층 등으로 구성된다. 본 고는 이 모델의 각 계층에 해당되는 추상적인 데이타 개념을 정의하고 이것이 각 계층의 서비스와 실재로 객체지향 기법으로 제공되는 API(Application Programming Interface)가 어떻게 연결되는지 기술하였다. 그리고 실제 구현하기 위한 구체적인 문제를 예를 들면서 다루었고 마지막에 추후 연구해야 할 방향을 제시하며 결론을 맺었다.

An Object-Oriented, Client-Server Architecture for a Generalized Multimedia Processing Model in a Distributed Multimedia System

DooHyun Kim[†] Young-Hwan Lim[††]

ABSTRACT

In this paper, we describe a multimedia data processing model that supports a wide variety of applications based on multimedia production model. This model supports network-transparent access to stored multimedia data, real-time multimedia input devices, and multimedia processing. The model addresses real-time data switching and delivery, as well as acquisition, processing, and output. Most translation, compression, and synchronization services are integrated. This model consists of three layers: (1) stream, (2) multimedia presentation, (3) hyperpresentation. This paper describes the data abstractions associated with each layer. These data abstractions provide a framework for defining the services provided by each layer, and describe the object-oriented mechanisms that provide those services. A sample scenario is presentated to illustrate the use of this model. A server-client architecture and implementation issues, and future directions are also discussed.

1. INTRODUCTION

We are developing a multimedia I/O system, called MuX. The MuX consists of a multimedia I/O server, a presentation manager, an application programmer's interface (API), and a scripting language. The MuX system provides real-time support for streams (e.g., audio, video, mouse, and graphics). It also provides the capability to interactively define, edit, and review a multimedia presentation via the API. Once a presentation has been edited, its structure can be stored for later retrieval, editing, and playback using the scripting language.

† 정 회 원 : 한국전자통신연구소 선임연구원
†† 정 회 원 : 한국전자통신연구소 책임연구원
논문접수 : 1995년 8월 1일, 심사완료 : 1995년 12월 8일

While a multimedia presentation is being presented, it can be controlled by either the application or the presentation manager. These multimedia presentations can be linked together to form hyperpresentations.

Most vendor multimedia solutions[33, 34, 35, 36] address the problem of attaching various media devices, such as VCRs and camcoders, directly to the computers. This approach to multimedia processing limits computers to being only I/O control boxes. Similarly, system developers and researchers have defined abstractions for multimedia processing that support specific vertical applications (e.g., computer-based training)[40], specific topics (e.g., synchronization) [15, 16, 37], or specific media types (e.g., music, video)[13, 29, 34]. This approach limits the richness of applications that can be supported by native services. A more general approach to multimedia computing is required to supports a variety of applications such as hypermedia[5, 22, 32], video conferencing[12, 30], multimedia authoring, archiving, and collaboration[6, 23].

To make multimedia technology more useful for information producer and hence for consumers, a multimedia production model was defined for the MuX. In the multimedia production model, multimedia information systems help users compose presentations by enabling them to record, process, mix, and integrate media from a variety of different sources, and store and play the resulting compositions. With an effective multimedia information system, a media producer can extract important information from stored media and compose a presentation that conveys a message concisely and effectively. Ultimately, a multimedia presentation will require less effort for a user to assimilate information, because multimedia presentations take advantage of the user's natural ability to process multiple information media in parallel, and thus magnify the impact and intelligibility of the information presented.

The objective of the MuX system is to provide a more conductive environment for the multimedia production model. To support the objective, the MuX provides the following functionalities:

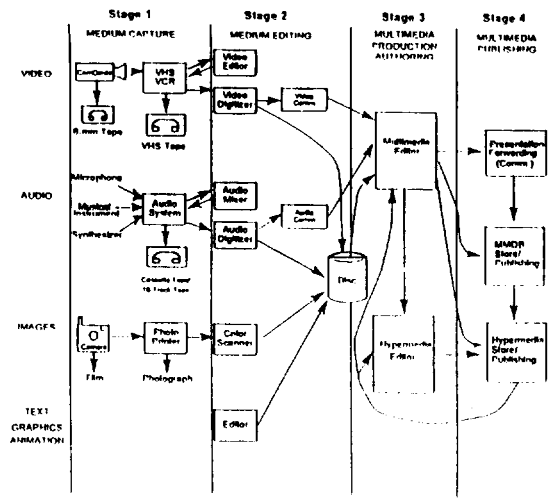
- o Media integration and syschronization
- o Adaptive, fine-frained media syschronization that adjusts to changes in resources and demand
- o Synchronization between the presentation of different media across multiple I/O channels
- o Network transparent access to multimedia data in a distributed system environment
- o Multimedia presentation creation, editing and control : the ability to define and modify presentation that specify time, space, and intermedia relationships
- o Powerful multimedia processing: capabilities such as
  - Data compression and decompression
  - Media transformation, including synthesis and decomposition, such as speech synthesis, speech recognition, natural language understanding, character recognition, and gesture recognition

This paper is focused on the multimedia data processing model and software architecture to embody the above functinalities, especially, the media integration and synchronization .

In the following section, we discuss the multimedia production, the multimedia processing model including its data abstractions, services and mechanisms. Then we discuss an object-oriented design issues and its implementation issues.

2. MULTIMEDIA PRODUCTION

In the multimedia production model, there are essentially four steps in preparing and using multimedia information, as illustrated in (Figure 1.)



(Figure 1) Multimedia Production Model Process

Step 1. Capturing multimedia data in real time: This step consists of capturing data from a single medium, such as video or audio, and recording it on some intermediate storage medium that may or may not be used as the final storage medium. Examples include shooting video on a camcorder using 8-mm tape, taping music on a cassette tape, or shooting a picture with a camera on 35-mm film.

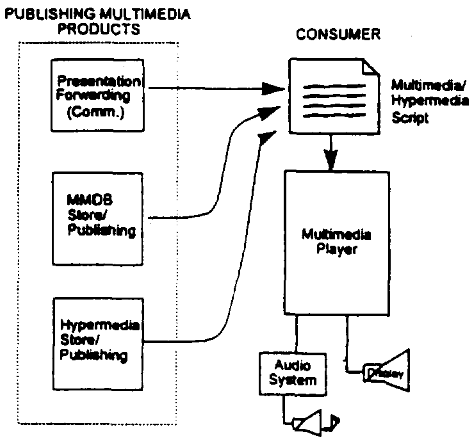
Step 2. Editing and representation of each medium's data: Depending upon the medium, this step consists of editing data from each medium in its original format (such as cutting and splicing video or filtering an audio recording); transferring medium from one storage format to another (such as transferring video from 8-mm tape to a digital format, or scanning a photograph); or actually creating data and editing the medium itself as in the case of text, graphics, or animations.

Step 3. Preparing a coherent multimedia presentation based on captured and edited data:This step consists of taking captured and edited multimedia data and integrating all of the data into a coherent presentation, including specification of the time and space relationships between data from each of the media, and creation of hypermedia links between multimedia presentations.

Step 4. Publishing the multimedia presentation: This final step is to store and prepare the delivery of the multimedia presentation. In preparation for delivery, the multimedia presentations are usually stored in a multimedia database; or some multiple presentations can be organized as a hypermedia presentation, which allows a user to browse interactively in a nonlinear fashion by traversing links and nodes. Finally, the presentation can be directly relayed to a communication channel for on-line presentation.

For real-time delivery of the multimedia information, some of the steps, such as Step 2 or Step 3, can be skipped;or the processed data can be passed directly to the next step without being stored. Depending on the application, these steps can be combined into a single process in the application. Multimedia teleconferencing, for example, does not have Step 2, so that after Step 1 the captured audio and/or video data passes directly to Step 3, in which this data is combined with other data such as mouse pointer input or text from a graphics window, and then is immediately broadcast to the other participant in the teleconference.

The consumer system, in contrast to the complex production system, will consist of only a multimedia browsing system. This browsing system will give end users access to on-line presentations via communication through the computer network, or by access to stored multimedia or hypermedia databases. Production and consumption of multimedia products are bridged by publishing multimedia products. The relationship between the published product and the consumer system is illustrated in (Figure 2).

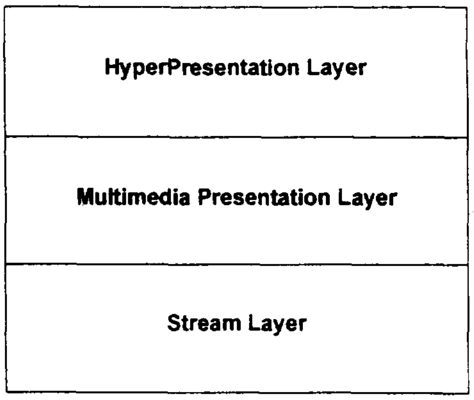

PUBLISHING MULTIMEDIA PRODUCTS
CONSUMER
Presentation Forwarding (Comm.)
MMDB Store/ Publishing
Hypermedia Store/ Publishing
Multimedia/ Hypermedia Script
Multimedia Player
Audio System
Display

(Figure 2) Relationship Between Published Multimedia Products and a Consumer

3. MuX MULTIMEDIA DATA PROCESSING MODEL

In this section, we present a multimedia data-processing model that is intuitive to developers and end-users of applications for multimedia production, but is powerful enough to accommodate real-time multimedia synchronization and integration services across a network of cooperative processors. This model supports network-transparent access to stored multimedia data, real-time multimedia input devices, and multimedia processing. The model addresses real-time multimedia data routing and delivery, as well as acquisition, processing and output. Media translation, compression, and synchronization services are integral to the model.

As illustrated in (Figure 3,) our model comprises a stream layer, a multimedia presentation layer, and a hypermedia presentation, or hyperpresentation layer. The stream layer provides services similar to a video router used in



(Figure 3) Multimedia Data Processing Model Layers

video production studios, where sources and destinations are local or remote files or devices (such as microphones, musical instruments, video cameras, displays, and audio speakers). The multimedia presentation layer is based on the concept of a programmable media multiplexer or media mixer, such as an audio mixer used in concert productions, or a switch or video editor used in a video production studio. In this layer, the media multiplexer takes input from a variety of sources, mixes them according to controllable parameters, and directs the result to an output port or destination. The hyperpresentation layer is a generalization of the links used in hypermedia document, where the document may include time-based media such as audio and video, and the links are dynamic and vary over time.

3.1 STREAM LAYER

The base layer of the model is the stream layer. The abstraction of a stream represents the data associated with a particular medium. Examples of media include standard media (audio, video, images, graphics, and text) as well as other media streams including mouse/keyboard, pen, animation, and musical instrument digital interface (MIDI) streams. These streams may originate from a file, a device, a connection, or from the higher layers. Streams representing these media can be classified into the following categories:

1. Digitally sampled continuous media streams: The medium stream represents a set of samples with a continuous sampling rate and pattern.
2. Synthesized continuous media streams: These streams are not originally generated by sampling a device. Rather, the output samples are synthesized from a data model to form a continuous stream.
3. Event-based streams: These streams are interrupt- or event-driven and therefore have a nondeterministic sample rate. Although the streams are not continuous, the stream data is time stamped at the time of each event. These streams often correlate to human interaction, such as mouse movement or keyboard input.

Streams within each of these categories can be further classified as real time (i.e., generated at the time of execution) or playback (i.e., prestored and played back from a storage device).

3.1.1 Stream Services

The services provided by the stream layer include the following:

○ Accessing multimedia data from a file, a device, or a connection, or from the higher layers.

○ Delivering multimedia data to a file, a device, or a connection, or to the higher layers, in a timely manner

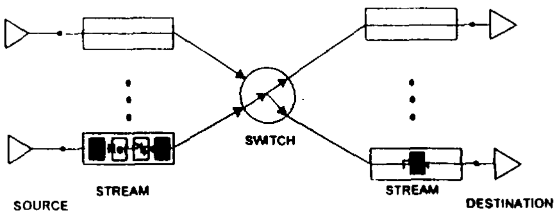
○ Processing of an individual stream (e.g., compression)

○ Selecting an input from one or more streams and distributing it to one or more destinations

○ Time-stamping or marking [Shepard 1990] stream data for downstream synchronization.

3.1.2 Stream Layer Abstractions

To provide these services, several abstractions and mechanisms have been defined, including a stream, source, destination, filter and filter pipe, and switch. These mechanisms are illustrated in (Figure 4) and are described in more detail below.



(Figure 4) Stream Layer Mechanisms

3.1.2.1 Stream

A stream is a flow of data through a conduit [Northcutt 1991] that reads data from a source, perform data type conversion, and deliver data to a destination. Source and destination mechanisms provide access to multimedia data in a file, device or connection. Sources and destinations are similar to transducers, described by Northcutt and Kuerman [1991]. Data from a source can be digitally sampled, synthesized, or event driven, as noted above. For synchronization purposes, the source is responsible for marking data or time stamping data with a system clock time value. For streams that originate from a remote site, the time stamp is corrected, within a margin of error, for differences between the remote site and the local site. A level of performance and quality of service between the destination and the sources may be specified for each stream.

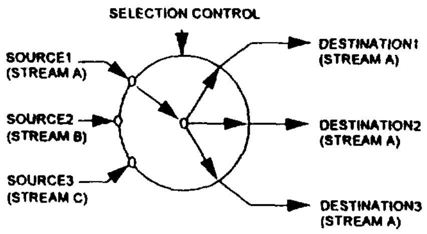
3.1.2.2 Filters*

Before a stream delivers data to a destination, a filter can perform one of several types of processing operations on it, including format conversion (e.g., RGB images to YUV images); data compression and decompression; and data type conversion (e.g., speech to text). Varying degrees of quality of service and performance can be achieved by having alternate filters for these operations.

The basic elements of a filter include an input, an output, control parameters, and a processing program. Filters can be combined to form filter pipes, or collections of filters. If a filter does not have any control parameters, or if the control parameters are provided at the time of processing (such as the quantization table for JPEG+ compression [Wallace 1991]), then it is said to be context free. A context-dependent filter operates within a context that can be specified and controlled independently of the data stream.

3.1.2.3 Switch

A stream switch provides two main functions: selecting an output stream from multiple input streams, and directing the selected media stream to multiple destinations. Selecting from multiple input streams can be used to support chalk-passing protocols [6]. Directing to multiple destinations allows the input stream to be tapped and tailored to form separate streams. AYUV input stream originating from a video device can be output to a stream that converts to an RGB video to be displayed locally and to an MPEG+ [13] compressed video stream to be sent to a remote location.



(Figure 5) Stream Switch

For example, consider the situation illustrated in (Figure 5). In this example, three streams (A, B and C) are all inputs to the stream switch. Using selection control, the switch can select one of the three input streams (in this case stream A). The selected stream is distributed to the output destinations. In this situation, streams B and C have no final destination, so, the switch may act as a destination and "stop" the flow from the corresponding sources.

* Although stream mechanisms are generally media independent, filter mechanims are, by nature, media dependent.

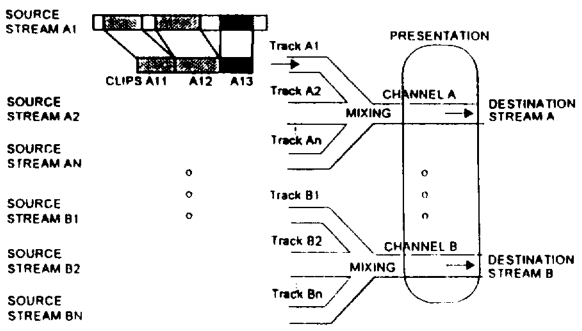
3.2 MULTIMEDIA PRESENTATION LAYER

The multimedia presentation layer builds on the stream layer. A multimedia presentation is a collection of streams that are coordinated with respect to time and space. Streams within a presentation are synchronized and have shared presentation control. There are logical groupings of media streams for integration and media-specific presentation control. The streams of dissimilar media (e.g., aural and visual) are synchronized and presented in parallel. Streams of similar media can be cut, reordered, processed, and mixed to form a new stream. These streams are grouped together as a channel for presentation or further processing. Example presentations are movies, videoconferencing, and collaborative work spaces.

The model's basic elements are derived from the video production model, but almost any media could be accommodated. Concepts from the video production model can be represented as follows: "tracks" are time-ordered streams; "channels" are a group of tracks associated with a mixer; and a "presentation" is the complete set of synchronized channels. The conceptual model for the multimedia presentation layer is illustrated in (Figure 6).

3.2.1 Multimedia Presentation Layer Services

The services provided by the multimedia presentation layer include the following:



(Figure 6) Media Integration Concept

ㅇ Interstream (i.e., parallel streams) and intra-stream (including integrated streams and single-stream order) synchronization [Yavatkar 1992]. Whereby the synchronization timing can be specified using hierarchical relationships, a time line, and reference points [Blakowski, Hubel, and Langrehr 1991].

ㅇ Integration of synchronized multimedia data, such as blending two video signals, mapping a video signal onto a graphics surface, and mixing multiple audio streams

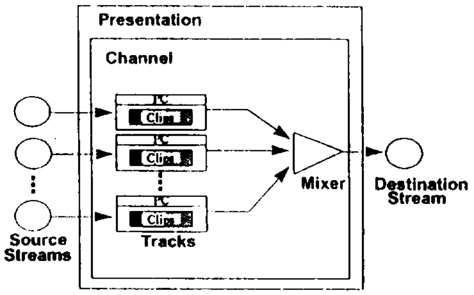
ㅇ Presentation-specific processing of a stream, such as chroma keying or warping a video signal.

3.2.2 Media Presentation Abstractions

To provide the media integration and synchronization services outlined above, we have defined the following mechanisms: logical time system, cue, presentation context, mixer, track, channel, and presentation. These mechanisms are illustrated in Figure 7 and are described below.

3.2.2.1 Logical Time System

As discussed earlier, a presentation is the central element for providing synchronization in a presentation. It interfaces between the timing mechanisms and the media integration mechanisms. The timing and synchronization mechanisms in the MuX server are based on the logical time system (LTS).



(Figure 7) Multimedia Presentation Mechanisms

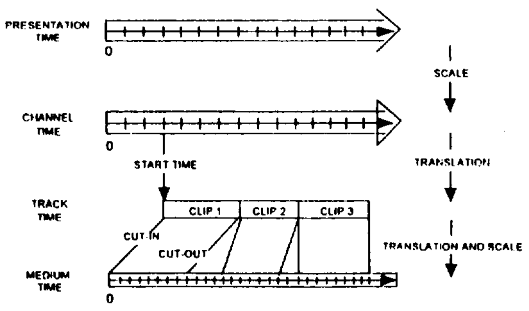
The LTS is a relative time system and consists of two major components: a start time and a tick interval. The tick interval is the time between ticks on the clock and is specified in a real-time measure, such as milliseconds. Given these two quantities, it is possible to transform between logical time and real time (i.e., absolute time).

An advantage of the logical time system is that time can be scaled simply by scaling the inner tick time [2, 24]. Another important aspect of the LTS is that it is possible to have different time domains for different elements. This allows significant flexibility.

We utilized this aspect to define the timing mechanisms for the MuX server based on the media integration concepts described earlier, namely, presentations, channels, tracks, clips, and the underlying media. The timing relationship between each of these components is illustrated in (Figure 8).

The presentation time domain in inherited from the master time domain. Channels inherit the time domain from the presentation. A track has a separate time domain derived by translating from the channel domain using the track start time. To get from the track domain to the medium time domain, both a translation and a scale must be performed. The translation is based on the cut-in time of the appropriate clip, and the scaling factor is a ratio of the established frame interval (i.e., the inner tick time of the master LTS) and the frame interval of the underlying medium. The scaling factor eliminates the problem of media with different frame rates. Note that by tying each of the channels to the master LTS time, synchronization between tracks can be achieved by simply specifying the track start time (in the presentation master LTS time domain) appropriately.

The logical time system (LTS) is the basic timing specification mechanism for synchronizing streams. Streams are synchronized by specifying their timing relative to the LTS. For example, channel times can be related to presentation time, and track times can be related to the channel times. Through the LTS, presentation of tracks, presentations, and channels can be scaled in a relative manner.



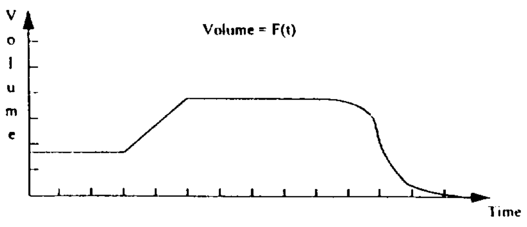
(Figure 8) Timing Relationship between Presentation Components

3.2.2.2 Cue

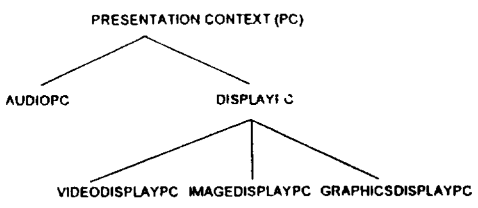
A cue is a synchronization mechanism that allows the specification of timing relationships between streams. It binds one timing event to another, e.g., the start of one stream is bound to the end of another. A timing event can be offset by a delta or a delay value. Given that the duration of the two related streams is known, complex temporal relationships can be specified using a cue; including before, after, meets, overlaps, starts, finishes, and equals [15].

3.2.2.3 Presentation Context

The presentation context (PC) provides the mechanisms to specify parameters that define how media streams are mixed, integrated, and



(Figure 9) Time-based Presentation Context



(Figure 10) Hierarchical Presentation Context presented to the end user. PCs are specific to the medium associated with a stream. The PC values can be specified as a function of time. For example, a PC for an audio stream would specify the volume at which that stream is presented. The volume can be specified to change over time as illustrated in (Figure 9). As shown in (Figure 10), there is one PC for audio media (APC), and there are three PCs for display media: video (VDPC), image (IDPC), and graphics (GDPC). The presentation context value associated with a PC varies over time, offering considerable flexibility in defining how media streams are integrated and mixed.

3.2.2.4 Mixer

The primary element for stream integration and intrastream synchronization is the mixer. Multiple integrated streams are synchronized at the mixer input. The mixer's main function is to take frames of data from the input streams and compose them into a single output stream. Internal to each mixer, streams can be hierarchically integrated [16, 24] where a group of streams is mixed with other streams, and so on.

The mixer defines a space or domain in which the media is composed. This domain provides a relative basis by which the individual streams can specify how they are integrated within that space. In a visual domain, still images, video images, and 2- and 3-D graphic objects are rendered to form a final rgbaZ image [7] using such techniques as warping, chromakeying, morphing [3], alpha blending, compositing, and texture mapping [8].

3.2.2.5 Track

A track mechanism facilitates the synchronization, reserialization, and media processing specification of a single-medium stream. Synchronization is achieved by specifying a track's start time relative to the logical time system of a channel. There are three timing-specification methods: (1) a priori specification of the track start time relative to the channel's time domain, (2) specification of track start time with a cue, and (3) dynamic direction from the application or user via a start command.

A track also reserializes streams through the use of clips. Clips define cut-in and cut-out (start and stop) times.

Each track has a presentation context associated with it. The presentation context is time based, relative to the track's LTS. When a track receives data from a stream, it converts the data's time stamp (based on real-world time) to logical time, and derives the presentation context values based on the logical time, to be used by the mixer for media integration.

3.2.2.6 Channel

A channel groups a set of tracks that are integrated by a mixer. It defines a common logical time domain for specifying the timing relationships of the set of streams being integrated. A channel also defines the interrelationship between the streams, including stream ordering and processing within a mixer. A channel has a time-based PC associated with it, which is applied in a hierarchical fashion to each of the tracks associated with the channel.

3.2.2.7 Presentation

A presentation groups together a set of channels that are synchronized in parallel. It provides a master time domain for specifying the time relationships of the presentation. As with the other mechanisms, it has a PC, which applies to all channels comprised in the presentation.

3.3 HYPERPRESENTATION LAYER

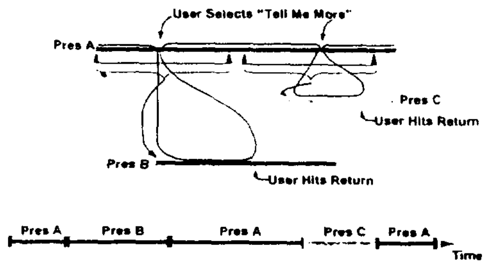
The highest level of the three-layer model is the hyperpresentation layer. The abstraction of the hyperpresentation layer is that of a dynamic network of multimedia presentations connected by links, forming multidimensional (time and space) multimedia presentations. Links that connect multimedia presentations together are dynamic in that they change over time and have fixed lifetimes during which they can be activated [Brondmo et al. 1990; Halasz 1988; Michon 1992; Ogawa et al. 1990; Zellwech 1992]. Users can interactively access and manage information contained within the hyperpresentation network by traversing links. In addition, related information can automatically be triggered for presentation when time-sensitive preconditions are met.

The infotmation content of a presentation changes over time. One portion, or segment, of a presentation may address one topic, while another segment may cover another topic. These segments may overlap. A specific point in time can be denoted by a milestone. Segments and milestones correspond to the visual or aural content (media specific), or correspond to all media that are being presented. Links between one presentation and other presentations come and go with their associated segments and milestones. For example, a segment of a presentation might generally cover "topic X"; and during the presentation of that segment a user

3.3.1 HyperPresentation Layer Services

The services provided by the hyperpresen-

might hit a "tell me more" button to see a related presentation on the same topic X. When finished with the side topic, the user could return to the original presentation. This type of interaction is illustrated in (Figure 11).



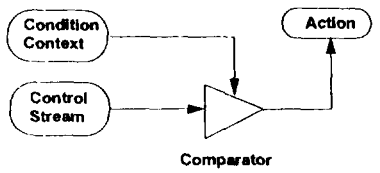
(Figure 11) Example Serial HyperPresentation

tation layer include the following:

- Time-based linking of multimedia presentations, based on the content of the presentation defined by milestones and segments that may overlap
- Triggering of parallel presentations when time-sensitive preconditions are met.

3.3.2 HyperPresentation Abstractions and Mechanisms

To provide theses services, the hyperpresentation layer supports the following mechanisms: conditions, control streams, actions, and comparators, shown in (Figure 12). Using these fundamental components, flexible, user-definable links and triggers can be constructed. Each of these mechanisms is described below.



(Figure 12) HyperPresentation Layer Mechanisms

3.3.2.1 Condition

A ondition describes an event upon which one presentation is associated or linked with another presentation. This condition is directly related to the information content of a segment or milestone of the originating presentation. The condition is only valid for the duration of the segment or instant of the milestone, which is defined relative to the logical time system of the originating presentation. A condition can be defined spatially with regions or hotspots [Michon 1992] that are compared to mouse input, aurally with tones or phrases that are compared to audio input, and gesturally with gestures compared to pen input, data glove input, or video input. The condition can also change over time. For example, a hot spot can move and change size along with a graphic object that is moving in a presentation. A presentation context associated with the condition defines how the condition is represented (e.g., visible or invisible button, ⟨x, y⟩ coordinate) to the user.

3.3.2.2 Control Stream

A control stream is the mechanism that can activate a link. A link is activated when a control stream meets a condition (defined above). A control stream can consist of, but is not limited to, input from the following: mouse, keyboard, data glove, video camera, or pen.

3.3.2.3 Comparator

A comparator is the primary mechanism for detecting a link or trigger condition and executing the actions necessary to link the presentations together. To perform this operation, a comparator has a control stream and a condition associated with it. During operation (between the start time and the end time), a comparator continuously compares the input from the control stream to the associated condition that activates a link or trigger. This implies that the clock or time system of the comparator must be synchronized with the clock or time system of the multimedia presentation. When a condition is met by the control stream, the comparator executes an action to link the presentations.

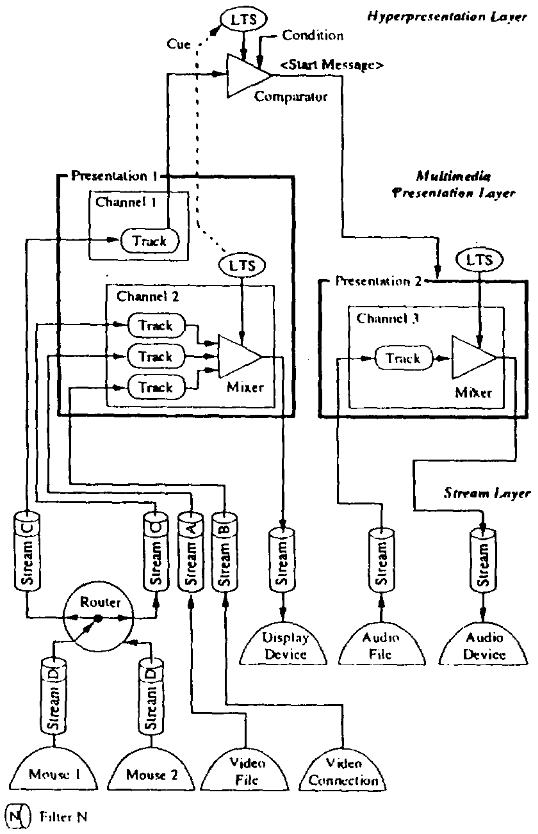
3.3.2.4 Action

When a comparator determines that a condition is met (e.g., a button was mouse selected at the location of a link), the comparator executes a set of programmatically defined actions that perform a link traversal or trigger. For example, if the semantic is a jump (i.e., a sequential link), the action pauses the current presentation, pushes that status of that presentation onto a stack, and plays a specified segment of the destination presentation, and upon returning from the destination presentation, resume the original presentation, using the status from the stack. Since the actions are defined programmatically, an almost limitless variety of links and triggers can be defined.

3.4 SAMPLE SCENARIO

A sample scenario, illustrated in Figure 13, shows how streams, switches, tracks, channels, presentations comparators, and compositors can be interconnected. This example demonstrates the manner in which tracks are aggregated into channels and channels are grouped into presentations, and how multiple presentations can interact through hyperlinks.

This scenario shows a collaborative hyperpresentation application in which two users manipulate independent mouse input devices. Their devices alternate in controlling the cursor through a user-defined chalk-passing protocol that controls the input selection of the switch. Two video sources are played in the first presentation. One video source is read from a file and the second comes from a network connection to a remote device. These streams request data from their input sources, process the data, and deliver to their respective tracks before their assigned deadline. The track converts the data's real-time timestamp into logical time and queues the data in a time-based queue until it is accessed by the mixer. At the appropriate time, the mixer accesses the data, using a synchronized data-extraction technique for both the video tracks and the mouse track (whose cursor icon is defined in the mouse track's presentation context). The mixer then integrates the data and writes the output to a stream that delivers the data to the output display device.



(Figure 13) Sample Scenario

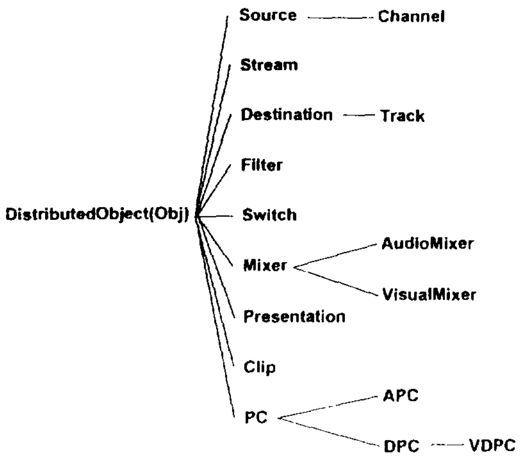
When the user with the cursor control (as determined by the switch) selects a hot spot, the comparator triggers a second presentation to begin presentation of an audio stream, which is played back from a sound file. To perform this operation, the comparator continuously checks conditions against the input control stream (in this case <x, y> coordinates of mouse key presses) against the location of the hot spot. When a condition stream matches user input, a trigger message is sent to the second presentation, notifying it to start playing at an indexed time into the presentation. This message propagates down to the channel, track and stream.

Filtering functions are illustrated in several places in the first presentation. Filter A for the video file input stream performs scaling of data, reducing the frame rate from 30 to 15 fps. The second video source, from the network connection, has filter B attached to its stream. Filter B decompresses data from MPEG encoding. Filtering is also performed on the input mouse streams. In filter C, the input events are transformed from global <x, y> coordinates into window coordinates. In filter D, events are propagated forward, based on whether or not they match the window id of the current focus.

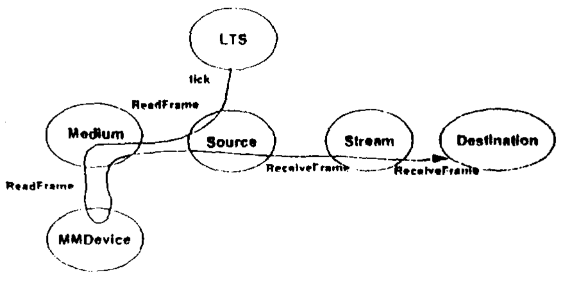
Timely delivery of data to the compositor, and from the compositor to the output destination, uses time constraints specified for the stream objects.

4. OBJECT-ORIENTED DESIGN

In our version of a prototype MuX multimedia I/O system, we have chosen to base our implementation on object-oriented design and implementation techniques. An object-oriented design provides good mechanisms for extensibility, maintainability, and portability. By providing general abstract superclasses, the system can easily be extended by adding subclasses. Extensions may include adding support for new medium data types, new filtering services, new mixing services, and new interfaces to devices. The modularity provided by using object-oriented design techniques also allow for isolation of device-dependent code, which simplifies porting tasks. We choose to use the C++ object-oriented language for our implementation. The class hierarchy of the base set of classes to support this model is defined in (Figure 14) based on the multimedia data processing model defined in Section 3. The classes illustrated in (Figure 14) are the classes that are exposed to a client application through the MuX client library. In addition to these classes, several other suites of support classes have also been implemented.



(Figure 14) MuX Multimedia Data Processing Model Class Hierarchy



(Figure 15) Basic Stream Layer Operation

4.1 Delivery and Routing Mechanisms

The Stream[1] is responsible for accessing data from Source and delivering the data to Destination. (Figure 15) illustrates this basic operation and shows the essential objects that perform the operations necessary to move data from Source to Destination. In order to access data and push it to a Stream, a Source consists of a collection of four objects: 1) a Source, the primary object; 2) an LTS that is used to provide timing events; 3) a Medium that provides a common interface to devices, and 4) a Multimedia Device, or MMDevice. In this scheme, the Source object acts as the interface to the other mechanisms. When a Source receives a Play message, it sends a Start message to the LTS, which forks a thread that is used to execute the tick operations. This thread immediately sets its deadline for the first tick of the clock and does a thread yield. Subsequently, the LTS thread returns from the yield and executes a callback that has been registered with the LTS for the tick operation. This callback function calls the calls Medium::ReadFrame( )[2] member function(Medium::ReadFrame( ) stands for ReadFrame member function of Medium class. The same notation is applied to the member functions of other classes). The Medium::ReadFrame( ) performs the device specific operations to read data from a multimedia device. This frame is returned to the Source object which then executes the Stream::receiveFrame( ) member function. In this basic operation example, the Stream::receiveFrame( ) simply passes the data on to the Destination via the Destination::receiveFrame( ) member function.

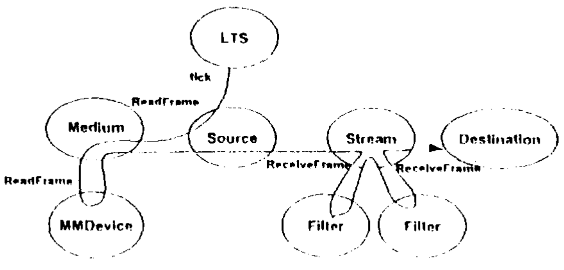
The abstract base class Medium is the basic interface elements used by all its subclasses and provides a common interface to

1) The capital 'S' of the "Stream" denotes the stream object. The same is applied to other objects.

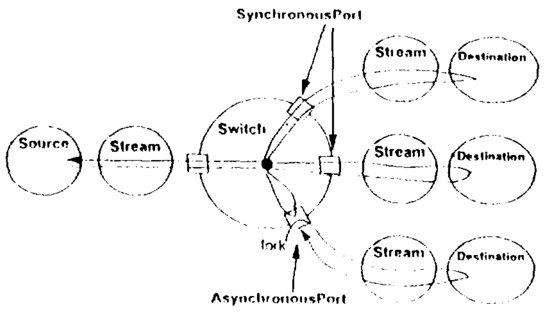
2) The "Medium::ReadFrame( )" denotes the ReadFrame( ) member function of a medium class. The same detational rule is applied to other member functions.

the Stream Layer. The Medium provides the device specific services required to read or write frame data.

In the case where filtering is applied to the data stream, the operation is much the same as described above. However, as illustrated in (Figure 16), the frame object is passed through a series of Filters that have been registered with the Stream object.



(Figure 16) Stream Layer Operation with Stream Filtering



(Figure 17) Stream Layer Operation with a Switch

In the case where a Switch is involved as illustrated in (Figure 17), the stream's data flow is split by the Switch object. The method for splitting the stream depends upon the type of Ports that the output Streams are connected to. If the Port is a synchronous Port, then the data is delivered to the Stream object using the same thread. If the Port is asynchronous, the a new thread is forked to deliver the data to the Stream. In either case, when the data buffer is delivered to the output Stream object the reference count is incremented rather than copying the data.

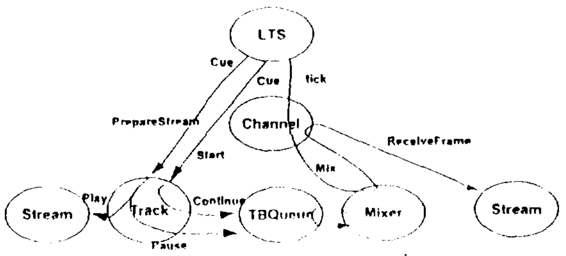
4.2 Synchronization and Media Integration Mechanisms

A multimedia presentation is a collection of streams that are coordinated with respect to time and space. Streams within a presentation are synchronized and have shared presentation control. There are logical groupings of media streams for integration and media-specific presentation control. The streams of dissimilar media (e.g., aural and visual) are synchronized and presented in parallel. Streams of similar media can be cut, reordered, processed, and mixed to form a new stream. These streams are grouped together as a channel for presentation or further processing.

4.2.1 End-Point Synchronization

The multimedia presentation layer elements consist of the following: "tracks" are time-ordered streams; "channels" are a group of tracks associated with a mixer; and a "presentation" is the complete set of synchronized channels. A Track object's main role is to connect a stream to a mixer and define the timing parameters necessary to mix and present a medium stream. To facilitate this, a Track essentially serializes a set of clips from a medium stream and queues it up for a Mixer to process. A Track has a start time that defines when the series of clips is accessed, mixed with other streams, and presented to the output stream. A Track also has an associated Presentation Context (PC), which defines how the medium should be composed and presented to the destination.

The Track's Presentation Context variables can have different values over time, thus allowing for such effects as fade-in or fade-out. A Track is also responsible for controlling the flow of data, including starting the stream and stopping the stream at the specified times. When the stream is flowing, between the start and stop times, the Track is also responsible for controlling the stream speed. This speed is defined by either the PC associated with the Track or as specified directly by the client.



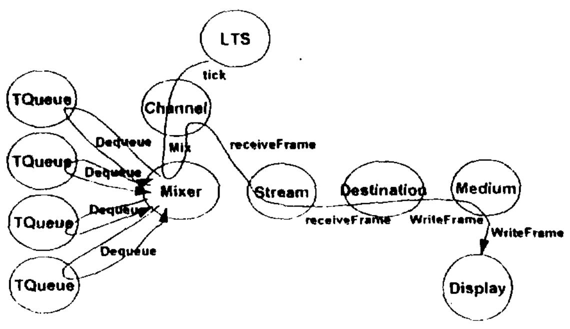
(Figure 18) Track and Channel Timing Mechanisms

To perform these operations, the Track registers three "actions" with the LTS clock bound to the Channel that the Track is associated with. As illustrated in (Figure 18), the first action is used to prestart the stream to account for access delay and ensure that data arrives when the start time arrives. The second action corresponds to the start time of the Track. The third action corresponds to the end time and is used to stop the stream. After receiving the first action callback and starting the stream, the Stream object delivers frames to the Track at a constant rate. When the Track receives a frame, it process the frame by converting the timestamp associated with the frame into Channel's time domain and enqueues it in the time-based queue for retrieval at a later time by the Mixer. In order to do this time conversion, it is necessary to know the real-time in which the track is started. This is the purpose of the second action callback. When this callback is received, the current time is captured and used for subsequent time conversions. Finally, the third action callback is used to stop the stream. In addition, when this call back is received the time-based queue is flushed to ensure that no residual frames are used.

4.2.2 Inter-Stream Synchronization

A channel is a logical construct that integrates a group of media into one logical I/O stream such as left audio channel (left ear) and right audio channel (right ear). To facilitate this integration, a channel bundles together a group of tracks that contain a series of clips from a medium stream. A channel integrates and synchronizes the composition and presentation of each of the tracks to a destination via the mixer. It defines a common logical time domain for specifying the timing relationships of the set of streams being integrated. A channel also defines the interrelationship between the streams, including stream ordering and processing within a mixer. A channel has a time-based presentation context associated with it, which specifies state variables and is applied in a hierarchical fashion to each of the tracks associated with the channel. In general, a Channel 1) groups tracks together, 2) directs tracks to the mixer, and 3) controls presentation of the integrated stream.

A Channel contains a PC which controls the presentation of the Channel's output. Since actual mixing of the data is done by the Mixer associated with the Channel, the Mixer is given a reference to the Channel's PC. This PC is associated with the root



(Figure 19) Channel Mixing Operation

group within the Mixer.

The Channel's LTS clock is used to control the synchronization of the tracks as well as control the mixing of the input streams and presentation of the result to the output stream. The start and stop times are controlled using "action"[3] events from the LTS (see Figure 18). The actual mixing of the streams is executed from a tick callback from the LTS. The process of mixing an individual frame is illustrated in Figure 19. When a tick callback is received from the LTS, the Channel executes a Mixer::Mix( ) member function call that performs the actual mixing of the stream. To execute the mix the Mixer in turn dequeues the appropriate frames from the TQueues associated with each track. After the frames have been collected, they are mixed together as defined by PCs and returned to the Channel. The Channel then delivers the mixed frame to the output stream by executing a Stream::ReceiveData( ) member function call. The output stream then proceeds to deliver the data to its final destination.

The Time-Base Queue (TQueue) provides a queue with some special and important properties. In addition to providing the typical enqueue and dequeue operations, a TQueue provides the ability enqueue and dequeue objects according to a logical time associated with the object. When a object is enqueued into the queue, a logical time is associated with the object, for example, 14. This object will not be dequeued unless an object of time 14 or greater is requested (assuming that the direction of the queue is "forward"). Further, for times which are greater than 14, the object associated with the time 14 will be returned from a dequeue request until another object that has a higher logical time than 14 is inserted, and less than or equal to the requested time.

The best way to describe this is through a simple illustration. For example purposes, let's say that objects are enqueued with the following sequences of logical times associated with them: 3, 4, 6, 9, 10, 11, 14. Given this sequence of enqueues, if dequeue requests are made for 1 through 14 in linear sequential fashion, the result would be the following: NULL, NULL, 3, 4, 4, 4, 6, 6, 6, 9, 10, 11, 11, 11, 14. This sequence may vary slightly depending upon the timing between when the objects are enqueued and when they are dequeued.

The important aspect about this queue is that it has the ability to smooth out jitter within the system and prevent drop-outs. These properties are particularly useful when the presentation parameters of the data objects may be changing even though some data object have been lost or dropped for various reasons.

4.2.3 Time Conversions

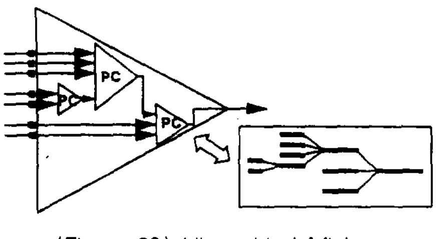
The MuX system handles four time domains, including logical time, real time, SMPTE time (an 80 bit field standard that defines time in hours : minutes : seconds. fra-

3) An action is a single timing event. At the time specified when the action is registered with the LTS, a callback is executed which effectuates the action.

mes), and UNIX time-val structure. It also provides operations for converting between the various formats. The synchron-ization begins at the source where data is marked [Salmony and Shephaerd 1990] or time stamped which is specified in real time, or UNIX time-val structure, or SMPTE time code. The time stamp is carried with the data to the point of synchronization. When data is exchanged between site, the time stamp is corrected, within a margin of error, relative to the site where the data is being synchronized. After a stream delivers the data to a track, the track mechanism converts the time stamp into logical time relative to its channel's logical time system and places the data in a time-based queue where beffering occurs.

4.2.4 Hierarchical Mixing for Media Integration

A Mixer is responsible for performing the mixing operations on a set of input streams in order to generate a single output stream. The Mixer class itself is intended to be an abstract superclass that is subclassed in order to support mixing of media specific streams, such as audio and visual streams. The Mixer base class provides several important functions, including 1) grouping of input Tracks together to form hierarchical relationships between the tracks, as illustrated in Figure 20; 2) associating a presentation context with

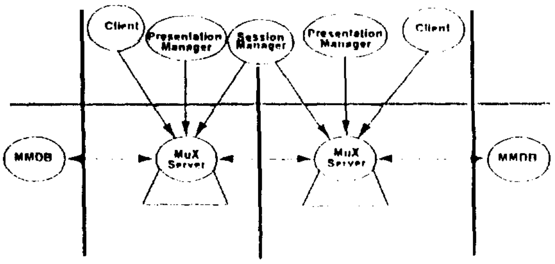


(Figure 20) Hierarchical Mixing

each of the groups that is used to control the mixing operations performed for each of the groups; and 3) collection of frames of data to perform mixing operations. The subclasses of the Mixer class perform the actual media specific mixing operation which is localized into the Mixer::Composite() member function. This member function is designed to be overridden by subclasses to the Mixer class.

5. MuX SYSTEM ARCHITECTURE

The general architecture of the MuX system is illustrated in (Figure 21). The central component of the MuX architecture is the MuX server. Media integration and synchronization, and multimedia processing services are provided by the multimedia I/O server. In addition, the server provides presentation control mechanisms; however, management and access to these mechanisms are provided by the presentation manager. In our approach, the presentation manager is a separate process from the server to allow tailorability and flexibility. Access to the capabilities of the server are provided by the multimedia application programmer's interface (API), which is implemented as a client library.



(Figure 21) MuX System Architecture Overview

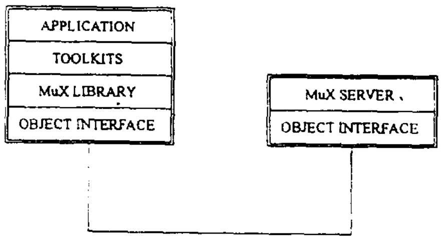
5.1 Virtual Object Interface

A very important aspect of the system is providing network transparent access to the services provided by the system, allowing applications to operate independent of the machine they are running on. The client-server approach to the system architecture provides the basis of this capability.

Since many of the multimedia applications being developed use an object-oriented approach to design and implementation, it is also desirable to provide an object oriented interface to the system. In an effort to combine network transparent service access and object-oriented programming the MuX architecture provides support for "virtual objects." Optimally an application would like to be able to communicate between objects with a minimum amount of programming effort. The simplest way to communicate between two objects is with a member function call in the same process address space. However, in a distributed environment where services may be located on other machines, this is not feasible and hence object messaging must occur across process boundaries. For a programmer, the concept of virtual objects simplifies inter-process object messaging significantly. With virtual objects, an proxy object interfaces between objects within one address space and the actual object which operates in another address space. The proxy object takes care of the interprocess communication aspects and hides the details such that the application programmer does not have to worry about it. To this goal, the MuX system employees distributed lightweight object messaging and utilizes this within the client-server domain.

The distributed object messaging services provide the ability to dispatch messages to appropriate objects within the network. To achieve this a run-time database of objects available within the system is maintained. When objects are created they are registered in the run-time database based on an assigned object id. Object messaging uses this id for object look-up, which is done via a directory service.

The MuX client library is layered on top of the distributed virtual object interface (Figure 22). This interface layer translates API messages into Object Message Protocol (OMP) packets. The encoded OMP packets are sent via an IPC mechanism to the MuX server, where they are translated into messages for server side objects.



(Figure 22) Distributed Object Messaging

The Object Message Protocol encodes messages and their parameters into packets that contain the object type, the identification code for the object instance, and the size of the data being passed to the server. The format of the OMP packet is shown in (Figure 23).

OBJECT CLASS	OBJ ID	MESSAGE	SIZE	DATA

(Figure 23) OMP Packet Format

5.2 Service Management

The objective of service management is to provide the ability to readily extend the beyond its initial capabilities, and allow the system to operate on multiple platforms with varying levels of support capabilities. Essentially, the basic system provides the core or essential set of "substrate" functionality, and "slots" allow the system to be extended. In this environment, non-core services are registered with the system providing extensibility, configurability, and portability.

Within the MuX system the types of services that we plan to support with service management include

- ○ Media handling services, including file handling and communication protocols
- ○ Device drivers or interfaces
- ○ Filters, and
- ○ Mixers.

5.3 Session Management

Session management provides the capability to control groups of streams and presentations in a coordinated fashion. Session management also facilitates external users or applications to join a session and utilize the information that is being exchanged within a session. These types of services should be provided at both the local and the global level. At the local level, streams and presentations are managed within a single station. The types of functions provided at the local level include managing the allocation of system resources to achieve the common objective of a group of streams or presentations, and "publishing" publicly available streams and presentations for use by remote applications. At the global level, streams and presentations are managed across multiple platforms. Global session management provide the ability of application to query what sessions are available at any given time and facilitate establishment of sessions.

5.4 Directory and Name Services

The directory and name services provided by the MuX system are primarily intended to be a database of run-time objects within the system. The objective is to provide the ability to register object in the database using the object id, a name or label, a class specific key, or an arbitrary key. After an object has been registered by one application, another application an look it up and utilize it's services. Additionally, applications can register with a directory to receive notification of changes to the object directory, specifying what types of changes they are interested in. We envision that the types of directories provided could include

- ○ Service directory of physical devices, filters and media services
- ○ Stream directory of currently active streams that an application could tap into and receive a split version of
- ○ Multimedia presentation directory of presentations that are active within the system
- ○ HyperPresentation directory of links and triggers that could be used for navigational purposes
- ○ Session directory of currently active sessions that would allow remote parties to join into a session.

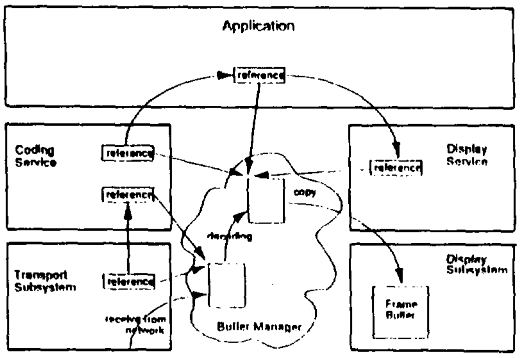
5.5 Presentation Management

Presentation management is also an important aspect of the MuX system. A presentation manager allows a user to adjust how media streams and multimedia presentations are being presented. This type of control can be applied at both the "master" level, adjusting the master volume and managing multimedia windows, and at the local level, controlling individual presentations, channels and tracks. The presentation management function could also allow users to select one media over another, providing focus control facilitated by dynamic resource management. The presentation management roles could also encompass adjusting system resource management parameters. This type of capabilities would provide users with the ability to tailor their environment to their specific needs or desires.

5.6 Buffer and Frame Management

Buffer management is a very important aspect of multimedia information systems due to the magnitude of the data that is being handled. To provide efficient management of data buffers, we implemented two object classes: a Buffer containing a block of data, and a BufferPool which manages a set of Buffers. There is only one BufferPool for the system. The BufferPool currently manages buffers or memory associated with the CPU, providing allocation and deallocation operations. In the future it will manage buffers through the system, including peripheral devices.

The BufferPool keeps track of the number of references to a Buffer. The actual location of the Buffer that contains the data is maintained by the BufferPool that allocates space for incoming data and manages the data as it is processed through the system, as illustrated in (Figure 24). With this scheme, data copying only occurs when it is written, such as in the decoding phase. Copying may also occur when the data is moved from main memory to device memory, such as frame buffer memory. With this technique only a reference to the data is passed through the system. When the number of references drops to zero, the Buffer is freed. This buffer management is a very important aspect for real-time processing as memory allocation and copying are expensive operations.



(Figure 24) Buffer Management within the System

Data is transported through streams in packets called frames. In the current implementation, a frame is a structure which contains a time stamp for the data, the format and type of the data, a pointer to the originating medium, a pointer to a PC for the data (see discussion of the Presentation Layer for details on the PC), and handles for the data and header information contained in the frame. The handles are references to control blocks in the BufferPool. The size and contents of the data and header blocks are unique to each format and type of frame. In most cases, these blocks of memory can be treated as opaque.

6. CONCLUSIONS

In this paper, we discussed a multimedia data processing model that supports a wide variety of applications based on multimedia production model. This model supports network-transparent access to stored multimedia data, real-time multimedia input devices, and multimedia processing. The model addresses real-time data switching and delivery, as well as acquisition, processing, and output. Most translation, compression, and synchronization services are integral to the model.

A fundamental difference between our model and others [Little 1990; Nicholaou 1990] is that our model is based on a full complement of generalised data abstractions of multimedia objects, namely streams, multimedia presentations, and hyperpresentations, rather than focusing primarily on data processing and synchronization. In our model, the data abstractions provide a framework for defining relevant and necessary processing and synchronization services and the mechanisms for providing those services. This leads to a model that is more intuitive to application developers and end-users, while still being powerful enough to accommodate real-time multimedia scheduling and integration services across a network of co-operative processors. A result of this approach is that much of the implementation details are hidden from the application developers, allowing them to focus on application-specific issues.

We believe the most of services provided by the model should be implemented at the system level on an object-oriented micr-okernel, with high-speed messaging to support the interactive real-time demands of time-critical media. Barring this ability, these services should be implemented via a single server located on each machine to provide integrated and central control of multimedia devices within a site, and efficient handling of multimedia data [Rennison et al. 1992]. Efficiciency considerations include support for copy-on-write and optimizations for efficient support data management, whereby the system provides real-time data flow control between buffers located on devices and in system memory. And also a best-effort real-time scheduler [Northcutt 1991] to provide timely acces, processing, and delivery of data to the compositors and devices should be supported by operating systems.

In the future, we plan to continue investigation into real-time communication support within the model, including multicasting in heterogeneous environments [Shacham 1992], Quality-of-Service control for parallel streams. And we also plan to specify primitive supports from operating system for the MuX system and to explore to incoporate these requirement into non real-time micro-kernels.

REFERENCES

[1] Accetta, M., R. Baron, d. Golub, R. Rashid, R. Tevanian, and M. Young. 1986. "Mach: A New Kernel Foundation for UNIX Development," Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh (August). Also in Proceedings of the Summer 1986 USENIX Conference, pp. 93-112 (July).

[2] Anderson, D.P., and G. Homsy. 1991. "A Continuous Media I/O Server and Its Synchronization Mechanism," Computer, Vol. 24, No. 10 (October).

[3] Beier, T., and S. Neely. 1992. "Feature-Based Image Metamorphosis," Computer Graphics, Vol.26. No. 2 (July).

[4] Blakowski, G., J. Hubel, and U. Langrehr. 1991. "Tools for Specifying and Executing Synchronized Multimedia Presentations," Second International Workshop on Networking and Operating System Support for Digital Audio and Video, Heidelberg, Germany (November).

[5] Brondmo, H.P., and G. Davenport. 1990. "Creating and viewing the Elastic Charles-a hypermedia journal," in Hypertext: State of the Art, R. McAleese and C. Green, eds., Intellect,

Ltd., Great Britain.

[6] Crowley, T., P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson. 1990. "MM Conf: An Infrastructure for Building Shared Multimedia Applications," Proceedings of the Conference on Computer-Supported Interactive Work, October 7-10 1990, Los Angeles, California, pp. 329 ff.

[7] Duff, T. 1985. "Compositing 3-D Rendered Images," Computer Graphics, Vol. 10. No.\x113(July).

[8] Foley, J.D., A. van Dam, S.K. Feiner, and J.F. Hughes. 1990. Computer Graphics Principles and Practice, Addison-Wesley, New York.

[9] Halasz, F. 1988. "Reflections on NoteCards: Seven Issues for the Next Hypermedia Systems," Communications of the ACM, Vol. 31, No. 7, pp. 836-852.

[12] Ishii, H., and N. Mikaye. 1991. "Toward an Open Shared Workspace: Computer and Video Fusion Approach of Team WorkStation," Communications of the ACM, Vol. 34, No. 12, pp. 36-50 (December).

[13] LeGall, D. 1991. "MPEG: A Video Compression Standard for Multimedia Applications," Communications of the ACM, Vol. 34, No. 4 (April). Leydekkers, P. 1991. "Synchronization of Multimedia Data Streams in Open Distributed Environments," Second International Workshop on Networking and Operating System Support for Digital Audio and Video, Heidelberg, Germany(November).

[14] Leydekkers, P, "Synchronization of Multimedia Data Streams in Open Distributed Environment," Second Intermational Workshop on Networking and Operating System Support for Digital Audio and Video, Heideberg, Germany, November 1991.

[15] Little, T.T.D., and A.Ghafoor. 1990. "Synchronization and Storage Models for Multimedia Objects," Journal on Selected Areas of Communications, Vol. 8, No. 3 (April).

[16] Little, T.T.D., and A.Ghafoor. 1991. "Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks," Computer, Vol. 24, No. 10 (October).

[17] Loeb, S. 1992. "Delivering Interactive Multimedia Documents over Networks,", IEEE Communications Magazine, Vol. 30, No. 5.

[18] Michon, B. "Highly Iconic Interfaces," in Multimedia Interface Design, Blattner, Meera M. and Dannenberg, Roger B., eds. ACM Press, 1992.

[19] Mullender, S. J., "Operating System Support for Distributed Multimedia," Usenix Summer Conference, 1995.

[20] Nicolaou, C., "An Architecture for Real-time Multimedia Communications Systems," IEEE Journal on Selected Areas of Communications, Vol. 8, No. 3, April 1990.

[21] Northcut, J.D., and E.M. Kuerman. 1991. "System Support for Time-Critical Applications," Second International Workshop on Networking and Operating System Support for Digital Audio and Video, Heidelberg, Germany (November).

[22] Ogawa, R., Harada, H., and A. Kameko. 1990. "Scenario-based hypermedia: A model and a system," in Hypertext: Concepts, Systems and Applications, A. Rizk, N. Streitz and J. Andre, eds., Cambridge University

Press, Great Britain.

[23] Poltrock, S., and J. Gudin. 1992. "Computer Supported Cooperative Work and Groupware," Tutorial Notes, CHI92, Monterey, California.

[24] Rennison, E., Rusti Baker., Doohyun Kim, and Young-Hwan Lim. 1992. "MuX: An X Co-Existent Time-Based Multimedia I/O Server," The X Resource, Issue 1, pp.213-33(Winter).

[25] Schwartz, F. 1992. "An Introduction to MiSC: Multimedia Glue for Bonding the Pieces," presented at the IEEE Microcomputer Conference, Asilomar, California (March).

[26] Shepard, P., and M. Salmony. 1990. "Extending OSI to Support Synchronization Required by Multimedia Applications," Computer Communications, Vol. 13, No. 7, pp. 399-406 (September).

[27] Steinmetz, R., "Synchronization Properties in Multimedia Systems," Journal on Selected Areas of Communications, Vol. 8, No. 3 (April) 1990.

[28] Stenmetz, R, "Analyzing the Multimedia Operating System," IEEE Multimedia, Spring 1995.

[29] Wallace, G.K. 1991. "The JPEG Still Picture Compression Standard," Communications of the ACM, Vol. 34, No. 4 (April).

[30] Watanabe, K., S. Sakata, K. Maeno, H. Fukuoka, and T. Ohmori. 1990. "Distributed Multiparty Desktop Conferencing System: MERMAID," Proceedings CSCW '90 Conf. on Computer-Supported Cooperative Work, Los Angeles, California, pp. 27-38 (October).

[31] Yavatkar, R. 1992. "Issues of Coordination and Temporal Synchronization in Multimedia Communication," Multimedia '92, Monterey, California (April).

[32] Zellweger, Polle T. 1992, "Toward a Model for Active Multimedia Documents," in Multimedia Interface Design, M. Blattner, Meera M. and R. Dannenberg, eds. ACM Press, U.S.A.

[33] Interactive Multimedia Association, "Multimedia System Services, Version 1.0," contributed by Hewlett-Packard Company, IBM Inc., and SunSoft Inc., June 1, 1993.

[34] Microsoft, "Multimedia Programmer's Reference for the Microsoft Windows Operating System," Microsoft Windows Software Development Kit, 1992

[35] Microsoft, "Digital Video Command Set for the Media Control Interface, Revision: 1.0," August 7, 1992

[36] Parallax Graphics, Inc., "XVideo Technical Overview Release 1.0," 1991.

[37] Ehley, L., B. Furht, and M. Ilyas, "Evaluation of Multimedia Synchronization Technique," Proc. of IEEE Int'l conference on Multimedia Computing and Systems, May 14-19, 1994, Boston, MA, USA, pp. 514-519.

[38] Arman, F., R. Depommier, A. Hsu, and M.-Y. Chiu, "Content Based Browsing of Video Sequence," Proc. of ACM Multimedia 94, Oct. 15-20, 1994, San Francisco, CA, USA, pp. 97-104.

[39] Mathur, A. G., A. Prakash, "Protocol for Integrated Audio and Shared Windows in Collaborative Systems," Proc. of ACM Multimedia 94, Oct. 15-20, 1994, San Francisco, CA, USA, pp. 381-388.

[40] Tobagi, F. A, "Distance Learning with Digital Video," IEEE Multimedia, pp. 90-93, Spring 1995.

김 두 현

1987년 한국과학기술원 전산학 졸업 (이학석사)
1985년 서울대학교 컴퓨터공학과 졸업 (공학사)
1987년~현재 한국전자통신연구소, 선임연구원
1991년~93년 스탠포드연구소 객원연구원

관심분야 : 분산 멀티미디어 처리, 멀티미디어 운영체제, 멀티미디어 통신프로토콜

임 영 환

1977년 경북대학교 수학과 졸업 (이학사)
1979년 한국과학기술원 전산학과 졸업(이학석사)
1985년 Northwestern University 졸업(이학박사)
1979년~현재 한국전자통신연구소 책임연구원

관심분야 : 멀티미디어, 초고속 정보 통신 시스템 소프트웨어, 에이전트