Implementing Multiplexing, Streaming, and Server Interaction for MPEG-4

Hari Kalva[†], Li Tang[†], Jean-François Huard[‡], George Tselikis[‡], Javier Zamora[‡], Lai-Tee Cheok[†], and Alexandros Eleftheriadis[†]

***Abstract* – We describe the implementation of a streaming client-server system for object-based audio-visual presentations in general and MPEG-4 content in particular. The system augments the MPEG-4 demonstration software implementation (IM1) for PCs by adding network-based operation with full support for the DMIF specification, a streaming PC-based server with DMIF support (via Xbind Inc.'s XDMIF suite), and multiplexing software. We describe *XDMIF,* the first reference implementation of the DMIF specification. The MPEG-4 server is designed for delivering object-based audio-visual presentation. We discuss the issues in the design and implementation of MPEG-4 servers. The system also implements a novel architecture for client-server interaction in object-based audio-visual presentations, using the mechanism of *command routes* and *command descriptors.* This new concept of command routes and command descriptors is useful in developing sophisticated interactive applications.**

***Index Terms* – MPEG-4, multimedia delivery, multiplexing, DMIF, interactive presentations.**

I. INTRODUCTION

Image and video encoding has been totally transformed with the advent of new coding and representation techniques [1][13][22][17]. These next generation coding techniques have made possible encoding and representation of audio-visual scenes with semantically meaningful objects. This new paradigm of object-based representation of audio-visual scenes and presentations will change the way audio-visual applications are created.

MPEG-4 is specifying tools to encode individual objects, compose presentations with objects, store these object based presentations and access these presentations in a distributed manner over networks. The MPEG-4 Systems specification [10] defines an architecture and tools to create audio-visual scenes from individual objects. The scene description and synchronization tools are the core of the systems specification. The scene description is encoded separately and is treated as another elementary bitstream. This separation allows for providing different QoS for scene description which has very low or no loss tolerance and media streams in the scene which are usually loss tolerant. The MPEG-4 scene description, also referred to as BIFS, is based on VRML and specifies the spatio-temporal composition of scenes. BIFS update commands can be used to create scenes that evolve over time. This architecture allows creation of complex scenes with potentially hundreds of objects. This calls for a high rate of establishment and release of numerous short-term transport channels with the appropriate Quality of Service (QOS). Traditional methods of signaling are not adequate to meet this demand because of the high overhead introduced. Furthermore, the applications should not depend on the underlying transport network. Delivery Multimedia Integration Framework (DMIF) is a general application and transport delivery framework specified by MPEG-4 [9]. DMIF's main purpose is to hide the details of the transport network from the user, as well as to ensure signaling and transport interoperability between end-systems. In order to keep the user unaware of underlying transport details, MPEG-4 defined an interface between user level applications and DMIF called the DMIF Application Interface (DAI). The DAI provides the required functionality for realizing multimedia applications with QOS support. While DMIF makes transport independent application development possible, there is also a need to develop transport dependent mappings for MPEG-4 content to be able to use existing infrastructure and support applications that do not use DMIF. The IETF's AVT group is specifying payload formats and synchronization schemes for delivering MPEG-4 presentations using RTP [5]. Since the existing infrastructure for digital TV uses MPEG-2 transport streams, to design MPEG-4 enhanced MPEG-2 content, techniques for the delivery of MPEG-4 over MPEG-2 transport are also being standardized [12].

We have developed an MPEG-4 system consisting of a client and a server communicating over an IP network using DMIF. The client is based on the IM1 player developed by the MPEG-4 systems group. We augmented the IM1 player [19] by adding a DAI to communicate with the server using DMIF. The client establishes a session with the server using DMIF signaling. The content/presentation to be played is communicated to the server during session setup. The flexibility of MPEG-4 while allowing interactive and complex presentations makes the content creation process non-trivial. Unlike MPEG-2, content creation process involves much more than multiplexing the media streams. Our experience with content creation has pointed out issues such as need for automation in the content creation process.

[†] with Columbia University, New York, [‡] with Xbind Inc., New York.

The MPEG-4 server, integrated with DMIF, delivers a requested presentation to the client. The presentations are stored in the form of Sync Layer (SL)Packetized Streams. These SL-Packets are time-stamped for a specific presentation and are not shared among presentations. We are working on a more efficient way of creating SL-Packets using the MPEG-4 file format. Creating SL headers just before delivery or re-stamping of packetized streams is necessary when elementary streams are manipulated, for example, stream control. The server uses the packet timestamps to time packet delivery so that receiver buffers neither under flow nor over flow.

The *XDMIF* implementation [23] we integrated is the first implementation of a DMIF compliant integrated system that supports different QOS requirements and the rapid creation of numerous transport channels and efficient allocation of network resources. The efficient allocation is partially based on the DMIF concept of a software-based channel multiplexer called FlexMux. The FlexMux entity is realized using the tools of the *xbind* broadband kernel, a programmable platform offering connection management, routing, QOS mapping and a choice of protocol stacks at connection set up time. *XDMIF* is the first reference implementation of the DMIF specification.

Even though MPEG-4 is meant for object-based interactive applications, the extent of interaction with objects is limited by the VRML interaction model that supports only local interaction and lacks important functionality such as content selection and stream control. While the DAI primitive to send user commands to the server has already been specified in DMIF, the syntax and semantics of the commands for user interaction is still being worked on. We are implementing the command descriptor framework that has been proposed to MPEG to complete the user interaction framework by supporting server interaction. Command descriptor framework is a flexible architecture that blends with BIFS and supports interaction with the server for functions such as content selection, stream control, and other application dependent scene interaction.

In this paper we describe our MPEG-4 system and discuss the issues in the design and development of such object based audio-visual systems. We describe the system architecture in Section 2. An MPEG-4 server is described in Section 3, client architecture and issues are discussed in Section 4, DMIF implementation and architecture in Section 5, and the command descriptor framework for application signaling is presented in Section 6.

II. System Architecture

The MPEG-4 system developed is an end-to-end system consisting of an MPEG-4 server, the DMIF component for signaling and session management, an IP network, and an MPEG-4 client for media playback/rendering. Figure 1 shows the components of the system. In this section, we describe the overall system operation. The individual components are described in detail in the subsequent sections.

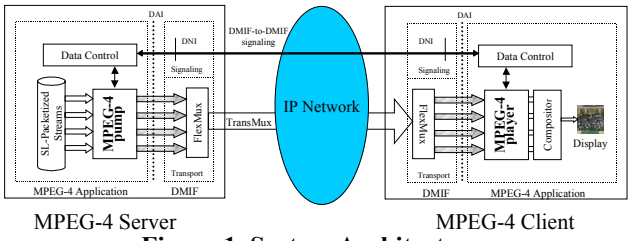

Figure 1. System Architecture

This system differs from the traditional video on demand systems in the characteristics of the presentations delivered. Video on demand has mainly been about delivering MPEG-2 audio and video [4][6][15][23]. In the case of object based presentations such as MPEG-4 presentations, the media data and the media composition data are transmitted to a client as separate streams, typically with different QoS requirements, in the same session. Furthermore, as the number of objects in a presentation can be quite large, the overhead required to manage the session is large. Interactivity makes this problem more complex as the resources required for a session will now depend on the user behavior, especially when user interaction with objects changes the number of objects in the scene either by adding or deleting objects. The MPEG-4 server consists of an MPEG-4, pump, an object scheduler, and a DMIF instance for IP networks. The server delivers Sync Layer Packets (SL-Packets) to the DMIF layer, which multiplexes them in a FlexMux and transmits them to the client.

The complexity of the player (i.e., client) has grown as a result of the new features and functionality offered by MPEG-4. The player is now also responsible for composing a scene from individual objects in addition to decoding and displaying the objects. A player consists of three logical components, a DMIF instance, elementary stream decoders, and a compositor. The DMIF instance is responsible for managing data access from a network or a file. A player typically contains several decoders each handling a specific elementary stream. Elementary streams are audio-visual streams as well as streams that describe the composition, rendering, and behavior of a presentation. Each object in a presentation is carried in a separate elementary stream. As MPEG-4 presentations can include media objects from several sources, potentially with different clock frequencies, there is additional burden on the client to track multiple clocks. This is typically done using soft Phase-Locked Loops (PLLs). Because of this additional complexity, a player's performance depends on the complexity of the content. Intelligent resource management and usage is necessary to use the resources such as memory efficiently.

The capability to add and remove objects during presentations and interacting with objects differentiates object based audio-visual presentations from traditional audio-visual presentations. DMIF supports this addition and removal of objects in a session efficiently. DMIF also supports network independence by providing a network independent API to the applications called, DMIF Application Interface (DAI). DMIF is also responsible for providing the requested QoS for the applications. Typically, streams with the same QoS requirements are multiplexed into a single channel called FlexMux. These FlexMux packets are transported to the other end on the under lying transport network where they are de-multiplexed by the peer DMIF entity and passed on to the application. DMIF is also responsible for communicating the user interaction commands from Command Descriptors by means of DAI user command primitives. These commands are transmitted over the DMIF-to-DMIF signaling channel. Our system uses a DMIF instance for IP networks and has been tested over Ethernet and satellite. The media streams are transported using UDP while signaling uses TCP.

A session is established before a server starts transmitting objects to a client. Session establishment is done by DMIF upon a request from the client. The MPEG-4 presentation to be delivered is selected by the client and communicated to the server as a part of the session establishment messages. MPEG-4 does not specify how a client selects a presentation. In our system, the client selects a presentation from a list of available presentations obtained by means of HTTP. The command descriptor architecture described in Section 6 provides a way to perform content selection using MPEG-4 scene description tools. As the session establishment continues, the server sends the initial object descriptor to the client. This initial object descriptor contains pointers to the elementary streams that are part of the session. The client uses this information to request additional channels for the elementary streams. Each presentation contains at least two elementary streams, a scene description stream and an object descriptor stream. In addition to these two elementary streams, a presentation may include a clock reference stream, a command descriptor stream, an Intellectual Property Management and Protection (IPMP) descriptor stream or an Object Content Information (OCI) stream in addition to the media streams. Inter-media synchronization is achieved using decoding and composition time stamps contained in SL-Packets. All elementary streams received by the client are time-stamped. The time stamp indicates the time an access unit is processed by the client. Each of the system components is detailed in subsequent sections.

III. MPEG-4 SERVER

Figure 2 shows the components of the MPEG-4 server. The server delivers objects in a presentation as scheduled by the scheduler. The elementary streams, carrying media and meta data for the presentation, are stored in the form of SL-Packetized Streams (SPS). The SL-Packet header contains information such as decoding and composition time stamps, clock references, and packet repetition flags. The scheduler uses the decoding timestamps to schedule the delivery of access units.

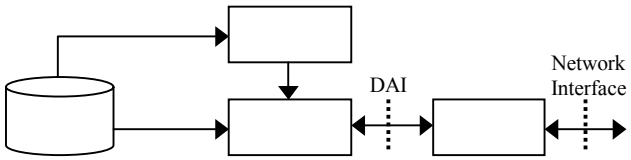

Figure 2. Server Components

A. Object Scheduler

Scheduling and Multiplexing of audio-visual objects in a presentation is a complex problem. The problem of scheduling audio-visual objects is similar to the problem of scheduling airplanes at hubs, bin packing, processor scheduling, and job shop scheduling. Because of the different application domain, there are no solutions that can be directly applied to the problem of scheduling audio-visual objects and also scheduling in the presence of user interaction that might alter the presentation. Scheduling of audio-visual objects has been the subject of study in [16], [20]. However these systems do not solve the problem of scheduling of audio-visual objects under resource constraints satisfactorily.

An MPEG-4 server that is transmitting objects should make sure that access units arrive at the terminal before their decoding time. The resource constraints that affect the transmission of access units are the channel capacity and buffer capacity at the receiving terminal. This problem has similarities with VBR scheduling, where the goal is to maximize the number of streams supported by a server. The difference is that in VBR scheduling discussed in [21], and references therein; the assumption is that the video data being handled is periodic (24 fps). In a general architecture such as MPEG-4, such an assumption is not valid as the data could consist of only still images and associated audio. In such cases the decoding times of individual access units have to be considered for efficient scheduling. Furthermore the delay tolerances and relative priorities of objects in an audio-visual presentation can be used to schedule objects for delivery. To satisfy the problem constraints, objects of lower priority could be dropped. Even different object-instances of an object may be assigned different priorities (e.g, I vs B frame). These characteristics of the audio-visual objects can be used by the scheduler in delivering objects.

The scheduler is also useful during the content creation process to determine if the presentation being designed can be scheduled for specific channel rates and client buffer capacity. It may not be possible to find a solution for a given set of resources; i.e, presentation cannot be scheduled with given resources. In order to create a schedulable presentation, some constraints may be relaxed. In the case of scheduling objects, relaxing a constraint may involve increasing the buffer capacity, increasing the channel capacity, not scheduling some object instances, or removing some objects from a presentation. We are developing

algorithms to determine the schedulability of a presentation during the content creation process.

It is also necessary to solve the problem on-line or in some cases computing incremental schedules. Computing a schedule in real-time is necessary to support interactive applications. When a user event adds a new object to the presentation, the resulting schedule has to be computed in real-time to determine if the event can be supported. A scheduler may also be used off-line to determine if an MPEG-4 presentation is suitable for a given channel and buffer capacities. In the current implementation, our scheduler uses an earliest deadline first (EDF) policy to schedule access units. EDF policy is optimal when buffer status at the terminal is not available.

B. MPEG-4 Pump

The MPEG-4 pump talks to a client via DMIF during session setup and delivers access units during the session. We are using a DMIF implementation called *XDMIF* [23] in our system. In this section we explain the session setup process using *XDMIF*.

To initiate a session, a client issues a DAI_Servic_Attach request to the server through *XDMIF*. The URL of the content that is to be played is passed to the server as a part of the DAI_Service_Attach message. The server responds by sending back the initial Object Descriptor (OD) of the content. The initial OD contains the list of ODs describing the elementary streams that are part of the session, the client decodes the initial OD and requests the establishment of data channels. The server maintains a list of sessions established with clients and a list of channels for each session. Session IDs and channel IDs are used to identify channels in signaling between clients and a server.

Upon service attachment, the server starts a new thread for the client. This thread is responsible for reading the SL-Packetized Streams and transmit access units to the client. MPEG-4 does not specify how a client receives the initial OD. In our system, it is done by including the initial OD in the response to DAI_Service_Attach request. When a client issues a DAI_Channel_Add request to receive an elementary stream, the DMIF layer at the server side, in conjunction with the pump, associates a channel number to the elementary stream and starts pumping the access units for that stream. Since a channel ID is just an arbitrary number assigned by *XDMIF* during channel establishment, the server also maintains a mapping between the channel IDs and ES IDs. This mapping allows the server to send a SL packet to the appropriate data channel through *XDMIF* to the client. The channel mapping is also useful to alter the data delivery on a particular channel in response to user interaction commands such as pause and resume.

Since the server uses a push model to send media data to the client, the data transfer should be properly paced to prevent buffer overflow or underflow at the client side. Buffer underflow will cause the interruption of media play. Overflow can cause data loss, which eventually cause impairments in the media played. We used send times derived by the scheduler based on the decoding timestamps in SL-Packets to pace the data transfer. Upon reading a packet from the file, the packet is either sent out immediately or held for an appropriate time period (i.e., the playing thread sleeps for a while) if the send time is later than the current time. The sleep time of the thread is determined by the difference of timestamps between the current packet and the previous one, subtracting the system time used for reading and transmission.

MPEG-4 does not specify how a client selects a presentation for playback. Currently, the only way that a client can request the content is to pass the URL of the content as the argument of DAI_Service_Attach function call. However, the DAI_Service_Attach is the first communication primitive between the client and the server. Therefore, the client has to have the knowledge of the available presentations on the server and select the content in advance. This is not a problem for local play back because a user can always access local disk and know what content files are available. In our system, clients establish a separate HTTP connection to access the content list maintained by the server. The server has to maintain the text file that contains the list and ensure the consistency of the list file and actual files available at the server side. More important, it is not flexible or interactive. We are currently implementing a content selection mechanism using the command descriptor approach described in Section 6.

Network delays and data loss could occur in the system if the content is not designed properly. A presentation created without the knowledge of target networks and clients could create long startup delays, and buffer overflows or underflows. This could cause distortion, hiccups in media playback and problems with the synchronization of different media streams. Consider a presentation that starts with several still images and starts a video and an audio stream five seconds after the beginning of the presentation. Since the presentation starts with still images, a higher bandwidth is necessary at the beginning of the presentation to transmit the images in time. If enough capacity is not available, it may not be possible to deliver the images before the video start time causing loss of synchronization. Unlike MPEG-1 and MPEG-2, MPEG-4 presentations are not constant bit rate presentations. The bit rate of a presentation may be highly variable depending on the objects used in the presentation. Furthermore, there is no notion of bit rate when images are used. These characteristics of MPEG-4 presentations make it difficult to design servers as well as content. A presentation may have to be recreated for different targets or servers have to be intelligent enough to scale a presentation for different networks/clients. Schedulers should be part of the content creation process to check the suitability of content for target networks and clients.

IV. MPEG-4 CLIENT

In order to achieve media synchronization when reconstructing the compressed audiovisual information, MPEG-4 defines a Systems Decoder Model, which abstracts the behavior of the receiving terminal in terms of synchronization, buffer management and timing for temporally accurate presentations of MPEG-4 scenes. These scenes, which are hierarchical groupings of media objects carrying syntactic or natural content, are composed using the scene description information coded in a binary format known as Binary Format for Scenes (BIFS). The BIFS is augmented by the Object Descriptor Framework. While the scene description declares the spatio-temporal relationship of the media objects, the object descriptors in the Object Descriptor Framework identify the resources for the elementary streams that carry these media objects and associate the streams with the media objects by means of Object Descriptor IDs. Together, the Systems Decoder Model, BIFS and Object Descriptor Framework form part of the basic building blocks for the architecture of the MPEG-4 client. In this section, we describe in brief one such existing architecture and then focus primarily on the discussion of issues related to the design of the client model.

A. MPEG-4 Client Architecture

Figure 3 illustrates the architecture for the implementation of an MPEG-4 client used in the streaming application. This implementation consists of the IM1 software [19] integrated with the XDMIF software. XDMIF provides signaling and transport functionality as well as an interface to the IM1 components. XDMIF is described in full detail in section 5. The IM1 software consists of several components, namely the controller, the SLManager (Sync Layer Manager) and the compositor.

The controller manages the flow of control and data information, the creation of buffers and decoders and attaches the transport channels established by the DMIF layer to the decoding buffers. The amount of buffer size to allocate for the decoding buffers is specified in decoder configuration descriptors.

The SLManager manages a set of transport channels, for receiving BIFS, OD and media streams, by binding them to their respective decoding buffers. Also it provides functionality for forwarding client requests, through the DAI, to the DMIF process and receiving both control and data information from the server. The compositor encompasses a set of other components for rendering MPEG-4 presentations/scenes and handling user events from the application.

First, the client application requests for a session establishment with the server and specifies the MPEG-4 presentation to be played. The SLManager forwards this request to the DMIF layer, which handles the establishment of the session. In case of a successful session establishment, the server provides the initial OD information. This information is passed from the client DMIF, through a specific DAI primitive, to the SLManager. The initial OD is used for allocating buffers for the scene description (BIFS), object descriptor and command descriptor streams.

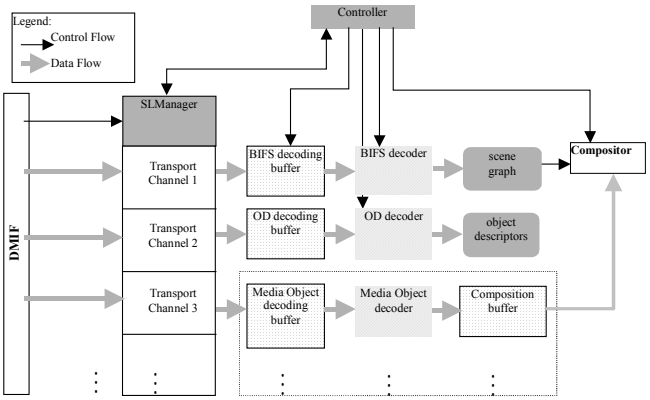

Figure 3. Architecture of an MPEG-4 Client

Then, the presentation controller, which as shown in figure 4 is a part of the compositor module, requests from the SLManager the establishment of transport channels for the reception of the associated media streams. The SLManager invokes the respective DAI command and forwards this request to the DMIF layer. Following, the DMIF establishes the transport channels taking into account the QoS characteristics of the media streams.

After the creation of the transport channels the client may start receiving the MPEG-4 data. The BIFS data units are decoded to form a scene graph, which is a hierarchical ordering of nodes that describe the media objects in a scene. The node attributes define the behavior and appearance of the objects as well as their placement in space and time. The OD data units are decoded into a list of object descriptors used for associating the media streams (i.e., elementary streams) with the media objects, and for configuring the system to receive media streams.

The decoded objects in an MPEG-4 presentation are composed and presented by the compositor. Figure 4 shows the architecture of a compositor, which consists of the presentation controller, an event handler and media object renderers. The presentation controller oversees the task of event handling and media object rendering.

There are two kinds of media objects that can be rendered: those that require retrieval of elementary stream data in the composition buffer (media stream objects such as video, audio, image), and those that can be rendered without reference to any elementary streams (non-media stream objects such as text, geometrical shapes like circles, rectangles, etc.). During media rendering, the data for the media stream objects are fetched from the composition buffers at times specified by the composition time stamps. The node attributes for the media stream object provides

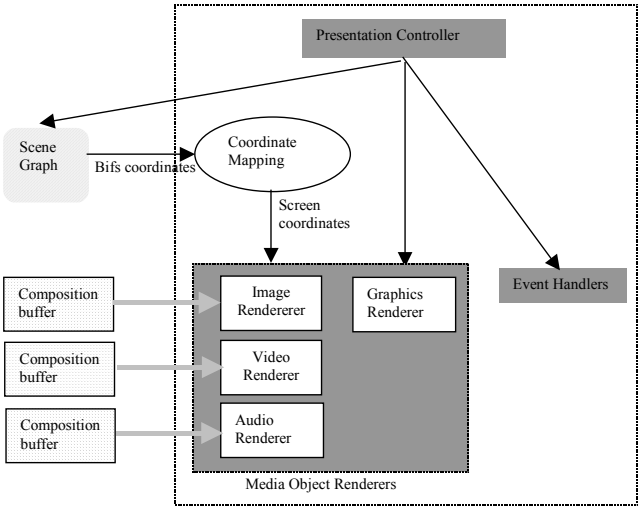

Figure 4. Architecture of a Compositor

information on the placement of the object within the scene. These are specified in BIFS coordinates and have to undergo coordinate mapping to derive corresponding screen coordinates to be passed to the image, video or audio renderers. Non-media stream objects also rely on information from the scene graph for the objects to be rendered by the graphics renderer.

Interaction behavior is specified in the scene description information encoded in the BIFS stream. This information specifies the modification of attributes of scene objects according to certain user actions. User actions on media objects (such as mouse clicks, etc.) are passed to the presentation controller, which then traverses the scene graph to derive the nodes of these media objects for firing the appropriate event handlers to process these events.

V. DMIF

A. DMIF Overview

The emergence of MPEG-4-based as well as other distributed multimedia applications exhibiting significantly more stringent QOS requirements than the conventional data-oriented applications imply the need for transport protocols with different characteristics to co-exist and be integrated within single applications. For the first time, application developers have to contend with the fact that a single protocol stack may perhaps be insufficient to meet all their needs. The same developers are also faced with the need of delivering their applications across multiple networks that use different transport and signaling technologies.

The solution to these problems lies in the definition of an application interface that hides all the transport details yet at the same time allows the user to express the needed QOS requirements in a simple manner. It is the role of the underlying system to map these QOS requirements into a suitable transport protocol stack and to provide an easy way of integrating new transport protocols without the need for modifying existing applications. The combination of a functional interface and a powerful underlying system eliminates the possibility of software investment being outdated because of new signaling and transport mechanisms.

In order to keep the user unaware of underlying transport details MPEG-4 defined an interface between user level applications and the underlying transport protocol called the DMIF Application Interface (DAI). The DAI provides the required functionality for realizing multimedia applications with QOS support. DMIF or the Delivery Multimedia Integration Framework is a general application and transport delivery framework. Specified by MPEG-4 [9], DMIF's main purpose is to hide the details of the transport network from the user, as well as to ensure signaling and transport interoperability between end-systems. Moreover, DMIF is expected to support the dynamic creation of transport channels with minimum signaling load and to utilize pre-allocated network resources in an efficient way.

Through the DAI the user may request service sessions and transport channels without concerns about the selected communication protocol. Furthermore, the DAI shields legacy applications from new delivery technologies since it is the responsibility of the underlying DMIF system to adapt to these new transport mechanisms.

DMIF also specifies an informative DMIF Network Interface (DNI) [9]. This interface highlights the actions that DMIF shall trigger with respect to the network, and the parameters that DMIF peers need to exchange across the network. The actions and parameters of DNI embody the semantics for DMIF Signaling (DS) Messages. DS Messages define a default session signaling protocol. DNI can also be mapped/piggy-backed on the existing signaling infrastructures.

Following, we describe the DMIF implementation used in our MPEG-4 system. This DMIF implementation is an ISO/IEC 14496-6 [9] compliant integrated system that supports different QOS requirements and the rapid creation of numerous transport channels and efficient allocation of network resources. The efficient allocation is partially based on the DMIF concept of a software-based channel multiplexer called FlexMux. The FlexMux entity is realized using the tools of the xbind broadband kernel [18] a programmable platform offering connection management, routing, QOS mapping and a choice of protocol stacks at connection set up time. The implementation of the DMIF layer using xbind is referred to as XDMIF and represents the first reference implementation of the DMIF specification.

B. The Model and Architecture of XDMIF

In this section, we present the system architecture of *XDMIF*, the first realization of DMIF. The architecture is based on the DMIF computational model that is described in the next section. The system architecture of *XDMIF* is discussed in section 5.2.3.

1) The DMIF Computation Model

The DMIF computation model is shown in figure 5 [9]. Two applications on the originating site request the establishment of sessions with two target applications. Two key sets of interfaces are identified. The first, the DMIF Applications Interface or DAI, provides an interface between the application and the transport system. The second, the DMIF Network Interface or DNI, is a semantic interface for the DMIF control plane that specifies the signaling messages and the parameters that DMIF peers need to exchange across the network, in order to establish transport channels. An end-to-end signaling communication path is established for the transfer of DNI messages. Figure 5 provides a high level view of a service activation and of the beginning of data exchange.

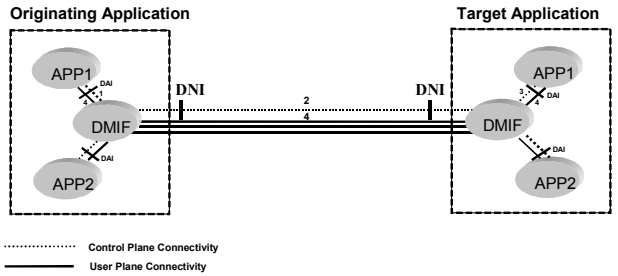

Figure 5. DMIF Computational Model

1. The originating application requests the activation of a service from its local DMIF layer by specifying the peer address and the name of the service managed by the remote peer. At this point a service session between the originating application and its local DMIF layer is created in the control plane. The service session is characterized by a unique service session identifier that the application will refer to in subsequent interactions.

The service session allows the setup of the MPEG-4 application (e.g., selection of decoder and decoding parameters). The decoder setup information is exchanged between the local and remote application through the DMIF layer by piggybacking application-to-application information in all DMIF signalling messages and by providing an opaque data structure as a parameter in most primitives of the DMIF application interface.

2. Through the DNI the originating DMIF contacts the target DMIF peer and creates a network session between them. In DMIF a network session is an association between two DMIF peers providing the capability to group together the resources needed for an instance of a service. All resources belonging to the same network session constitute the scope of the session. During the establishment of the network session, data needed for setting up the MPEG-4 application is exchanged. The information returned by the peer application is kept in the DMIF layer for future use (e.g., when a new service request is made, the DMIF layer immediately responds to the request without going through a new network session establishment phase). Network sessions have network-wide significance, whereas service sessions have local meaning. The association between these sessions is maintained in the DMIF layer.

3. The target DMIF peer identifies the target application and forwards the service activation request. As in step 1 a service session between the target application and its local DMIF layer is established in the control plane.

4. In the context of a particular service session the peer applications request the creation of transport channels for the transfer of MPEG-4 streams. In figure 5, the signaling message flow for the transport channels establishment is through the 1, 2 and 3 communication paths, while the transport channels are indicated by number 4. The established channels will carry, in the user plane, the actual information exchanged by the applications. Each transport channel belongs to the network session previously established.

The DMIF model provides unprecedented flexibility in terms of the number of transport channels, their bandwidth and QOS requirements. Such transport channels might be established across different networks based on different networking technologies. In short, DMIF makes the networking technology transparent to the application.

2) XDMIF *System Architecture*

Figure 6 depicts the systems architecture of XDMIF and illustrates some of the interactions and components involved during the creation of a multimedia service. The architecture closely follows the DMIF computation model depicted in figure 5. The system architecture consists of a signaling and a transport component. These entities or components support the function calls of the DAI. The C-Plane component might interact with a resource manager that establishes connections and allocates network resources for transport and/or signaling channels. In the case of connection oriented networks the latter role is played by a connection manager (CM). Figure 6 also depicts the FlexMux and TransMux components of the U-Plane.

The user plane DMIF layer is divided into Flexible Multiplexing (FlexMux) and Transport Multiplexing (TransMux) sub-layers. The FlexMux entity models a multiplexing layer which allows grouping of ESs with similar QOS requirements, while the TransMux Layer models the protocol stack that offers transport services matching the requested QOS.

The FlexMux represents a multiplexing layer that provides interleaving of one or more SL-Packetized Streams belonging to different ESs with similar QOS requirements. Each SL-Packet is mapped into a specific FlexMux channel and is provided to the FlexMux layer through the DAI. The FlexMux layer encapsulates one or more SL-Packet(s) into one FlexMux-PDU. The sequence of FlexMux-PDUs interleaved into one stream is called the FlexMux stream. The FlexMux demultiplexer retrieves SPSs from the FlexMux streams and through the DAI invocations provides them to the application.

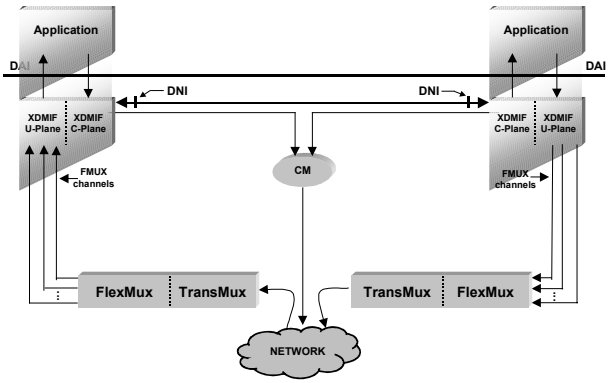

Figure 6. XDMIF System Architecture

The FlexMux layer does not provide a reliable error detection or framing of its streams. This functionality is provided via the underlying TransMux layer. The generic term TransMux layer is used to abstract any transport protocol stack that is suitable to transport MPEG-4 data streams and fulfill the end-to-end QOS requirements. The selected transport stack should provide appropriate framing of FlexMux streams, error detection and, if possible, delivery of corrupted data along with an error indication.

The realization of DMIF shown in figure 6 is within the framework of the xbind broadband kernel. The architecture allows for the dynamic creation of protocol stacks by binding transport components at run-time [7]. It further allows multiple implementations of a protocol stack layer to co-exist within a single application and for new capabilities to be added without having to modify any of the existing architectural components.

The xbind broadband kernel is an open programmable networking platform for creating, deploying, and managing sophisticated next-generation multimedia services. It is conceptually based on the XRM – a reference model for multimedia networks (see [18] for details). The term 'broadband kernel' is deliberately used to draw analogy to the operating system-like functionality that the system must support, namely one of a resource allocator and an extended machine.

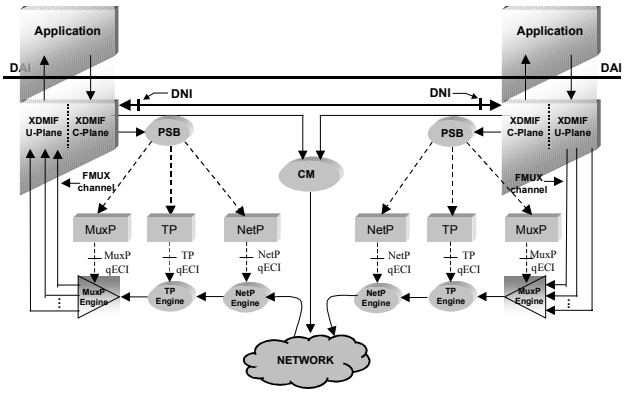

Figure 7. XDMIF System Architecture with Programmable Transport

In the next two sections we will describe the two key components of the XDMIF architecture. We will start in section 5.3 with a detailed presentation of the implementation of the C-Plane of XDMIF. This will be followed in section 5.4 with a brief presentation of the functionality of the architectural components of the U-Plane.

C. *Realizing the* XDMIF *C-Plane*

The programming model of the DMIF layer is object oriented. Each network session between DMIF peers is handled by a network session object identified with a unique end-to-end network session identifier <ns_id>. As shown in figure 5 more than one service session may be supported within the same network session. Each service session is modeled by an object. Service session objects are identified by a unique <service_id>. Within a service session, transport channels with specific QOS characteristics are represented. Each transport channel is handled by a TMux object. TMux objects are identified with a unique TransMux Association Tag (TAT).

According to the DMIF FlexMux concept, data originating from different ESs may be multiplexed into the same transport channel. Therefore, a TMux object may contain information about multiple FMux channels. Each one of them is handled by a FMux object and distinguished by a unique Channel Association Tag (CAT). Within a network session each TMux object is retrieved by the <ns_id, service_id, TAT> triplet, while each FMux object by the <ns_id, service_id, TAT, CAT> quadruplet.

The role of the DAI is to hide the transport details from the application by allowing the user to specify the QOS requirements for the transported streams and translating the user requests, across the DAI, into respective DNI signaling messages. The DAI is comprised of the following classes of primitives:

- Service primitives, which deal with the Control Plane, and allow the management of service sessions (attach and detach);
- Channel primitives, which deal with the Control Plane, and allow the management of transport channels (add and delete);
- Data primitives, which deal with the User Plane, and serve the purpose of transferring data through channels.

The implementation of the DMIF C-Plane is object-oriented. Each of its key elements is implemented as a C++ class. At runtime, the elements' interface provides functions for controlling and updating the state of the referred element instance. For example, each instance of a transport channel keeps information regarding the channel, e.g., QOS, allocated bandwidth and the destination address.

For the communication between an application and the DMIF C-Plane two abstract C++ classes are provided [8]. The first class is used for communicating between the application and the DMIF layer and implements the DAI primitives with a _Req and _Rsp extension. The second class is used for communicating between the DMIF layer and the application and implements the DAI primitives with a _Ind and _Cnf extension.

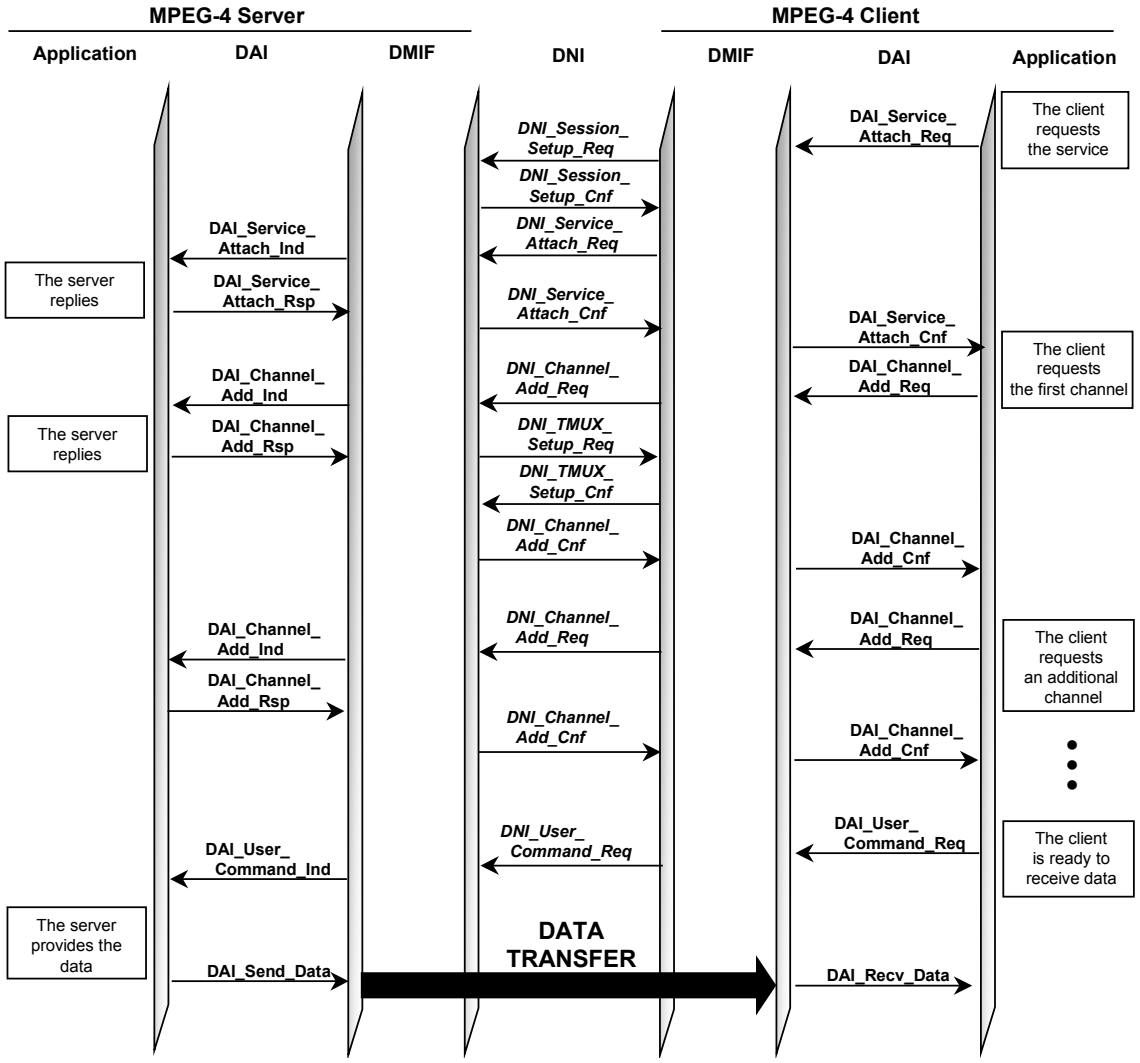

Figure 8. DMIF Message Flow

The XDMIF C-Plane is implemented as a 32-bit library by using the Microsoft VisualC++ compiler (version 5.0) and runs under the Microsoft Windows NT 4.0 operating system. The reader might find it instructive to download a demonstration version of the software from [23].

D. Realizing the XDMIF *U-Plane*

A wide variety of different protocol stacks (e.g., UDP/IP, AAL5/ATM, RTP/UDP/IP, H.223/PSTN, etc.) should be mapped into FlexMux and TransMux layers. How can all these layers or modes of communications be supported in a transparent way to the applications? How will the QOS requirements be provided? What is the complexity of the implementation? The answer lies in using the programmable transport architecture of the xbind broadband kernel [7].

Figure 7 shows the programmable component of the transport architecture of XDMIF. The architecture consists of a protocol stack builder (PSB) and a set of consumer/producers modeling the FlexMux and TransMux components shown in figure 7.

The consumers/producers in figure 7 are separately represented by their transport abstraction, called an engine, and their control and management abstractions, called front-ends. The consumer/producer engines (CPGs) are responsible for data processing (including protocol implementation), buffering and multiplexing. CPGs are located in the data path of the multimedia transport channel and are the actual hardware and software components that process and transport the data. Examples of consumer/producers are the NetP engine (e.g., IP, ATM, etc.) and NetP, the TP engine (e.g., TCP, RTP, etc.) and TP, and, the MuxP engine (e.g. FlexMux) and MuxP, respectively. By directly acting on the latter abstractions, the PSB controls the generation, consumption and processing of streams without being located in the data path [7], [8].

As shown in figure 7, the FlexMux functionality is provided by the XDMIF MuxP component while the TransMux layer corresponds to both TP and NetP components. Each transport channel is represented by a TMux object

identified with a unique TransMux Association Tag. The transport channel is controlled by the TP and NetP XDMIF components. The FMux objects are controlled by the MuxP XDMIF component.

The protocol stack is modeled in an object-oriented manner. The protocol stack builder is responsible for the construction of the protocol stack by binding MuxP, TP and NetP together. It is aware of a variety of transport components supported on the end-systems and the order in which they have to be bound to create a useful protocol stack. The PSB performs the dynamic binding of the components at run-time and creates a suitable transport protocol stack according to application QOS requirements.

The PSB dynamically creates a variety of protocol stacks on a per call basis that are tailored to the special needs of the application. This differs significantly from the traditional transport architecture that assumes pre-installed transport protocol stacks that cannot be customized.

E. DMIF Message Flow

Figure 8 shows the DMIF message flow between client and sever in our MPEG-4 system. This flow includes the primitives across the DAI as well as the corresponding DS messages through the DNI. First, the client application requests the establishment of a service session by calling the DAI_Service_Attach_Req() function and passing as parameter the DMIF-URL address of the peer entity. If there is no active network session with the peer, a signaling channel is established. For the purposes of this application, the signaling channel is a TCP channel and carries exclusively the DS messages. This channel also represents the network session between the DMIF peers. The DAI_Service_Attach_Cnf() indicates the end of this phase and passes to the application the service session identifier [ss_id] that identifies the session in the application level.

Second, the client checks the received ES descriptors and asks for the establishment of transport connections for the delivery of desired media streams. In our MPEG-4 system, one TransMux channel is adequate to convey the media streams. For the IP case the established TransMux transport channel is a UDP socket. The FlexMux component is used and 10 logical FlexMux channels are created within the TransMux channel. Each one of them is identified in the DMIF level by a unique [CAT] value and in the application level by a unique [ch_id] value. The [ch_id] value is provided to the application through the DAI_Channel_Add_Cnf() function after each channel establishment completion.

Then, the client calls the DAI_User_Command_Req() function to notify the server that it may start sending data in the indicated channel. The server through the DAI_Send_Data() function starts transmitting the media streams while the client DMIF process through the DAI_Received_Data() function forwards them to the application. The same sequence is repeated for all the established channels and the received MPEG-4 streams are presented on the screen.

VI. APPLICATION SIGNALING AND USER INTERACTION

Interactivity in the case of video on demand systems mainly meant supporting VCR functionality in the system. There has been some work on supporting this functionality. DSM-CC and RTSP are designed to support stream control in audio and video delivery over networks. With object-based video, interactivity means different things depending on the application and the media types involved. Object based representation of audio-visual objects and scenes lends itself to the design of more sophisticated user interaction with scenes and objects in the scenes. The user interaction mechanisms developed for video-on-demand are not sufficient to support user interaction in object-based AV applications as they were primarily designed to support stream control. In this section we present an architecture that supports user interaction in MPEG-4 systems. This architecture, called CommandDescriptor architecture, integrates well with the MPEG-4 scene description framework and supports fully interactive applications.

A. MPEG-4 User Interaction Model

Interaction with objects in an MPEG-4 presentation is supported by the VRML interaction model. An application's response to user interaction is determined by the content creator. Clicking on an object might cause the object to move in one application and hide the object in another application. The interaction behavior of objects cannot be standardized but VRML model assures consistent interaction behavior across clients and implementations. However, with VRML interaction model, all interaction is local to the client. This severely limits the interactivity in the system. The ability to interact with a server is necessary in many applications. For simple functions such as stream control, RTSP or DSM-CC user commands may be sufficient. In an object-based system like MPEG-4, interactivity is more than stream control and is very much application dependent. Since defining an exhaustive list of user interaction messages and system's responses is impossible, a generalized model that supports complete application dependent interactivity is necessary. The Command Descriptor framework, currently in MPEG-4 Systems version 2 VM, has all the features to support server interaction in MPEG-4 systems[1].

B. Command Descriptor Framework

The Command Descriptor framework provides a means to associate commands with media nodes in a scene graph. When a user interacts with the media nodes, the associated commands are processed. The actual interaction itself is specified by the content creator and may be a mouse click or mouse-over or some other form of interaction. The command descriptor framework consists of three elements, a CommandDescriptor, a CommandROUTE and a Command. Command descriptor, as the name implies, describes a user interaction command. Command descriptor

[1] A modified version of the proposal has since been adopted by the MPEG-4 Systems group [1][11].

are attached to media nodes via command ROUTEs. When a user interacts with a node, the associated command descriptors are processed. The processing simply consists of passing the command back to the server over a user command channel, or it may also involve modifying the command parameters before sending the command to the server. More than one command descriptor may be attached to a node and in that case they are all processed when a user interacts with that node.

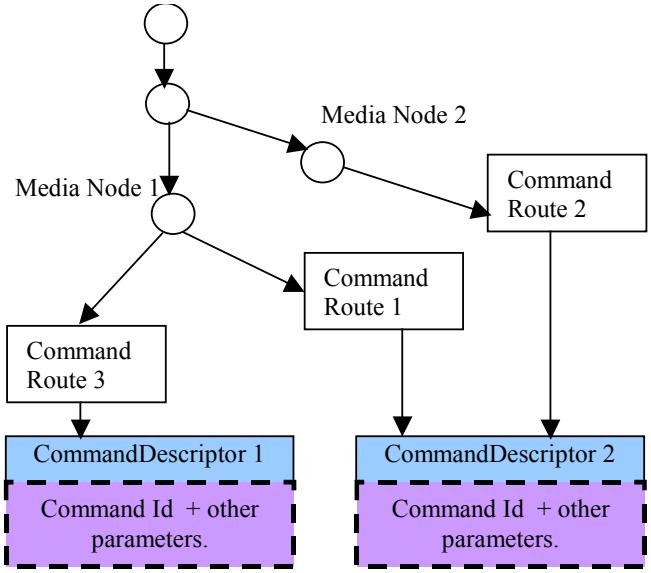

Figure 9. Command Descriptors and Command ROUTEs

Figure 9 shows an example with command descriptors attached to media nodes. Two nodes may share a command descriptor if the interaction with the two objects results in identical behavior. Command descriptors are transmitted to the client in a separate elementary stream called command descriptor stream. The command routes are similar to the ROUTEs used in scene description but point to a command descriptor. The command routes are included in the scene graph description and are transmitted in the scene description stream. This separation between commands and command association allows us to modify command syntax without editing a scene graph. Figure 10 shows the elementary streams involved in a system supporting server interaction.

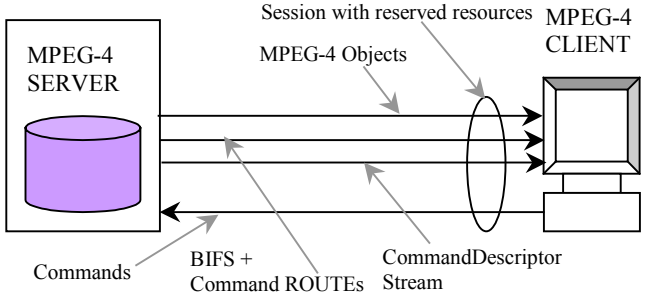

Figure 10. Elementary Streams in an MPEG-4 System Supporting User Interaction.

The command descriptors contain a command that determines the interaction behavior of the associated node. Command descriptors can be updated using command descriptor updates allowing a node to support different interaction behavior at different times depending on the command in the command descriptor. The commands are transmitted to the server using the DAI user command primitives supported by DMIF.

Even though command descriptor framework is generic, and supports application specific user interaction, we need a few standard commands to support consistent application behavior across applications and servers especially for common commands such as stream control. We specify a set of common commands, e.g., stream control commands [14] while the content creator specifies application specific behavior. A server should be aware of these application specific commands in order to process them.

```
class CommandDescriptor:
        bit(8) commandDescriptorTag = 0x06 {

    bit(16) CommandDescriptorID;

    bit(16) CommandID;

    // stream count; number of ES_IDs
    // associated with this message
    unsigned int (8) count;

    // ES_Id (channel numbers) of the streams
    // affected by the command
    unsigned int (16) ES_ID[count];

    // application-defined parameters
    do {
        unsigned int (8) paramLength;
        char (8) commandParam [paramLength];
    } while (paramLength!=0);
}
```

Figure 11. Command Descriptor Syntax

Figure 11 shows the syntax of a CommandDescriptor. Command descriptor ID uniquely identifies a command descriptor in a presentation. The command ID indicates the semantics of the command. This simple structure along with command routes in the scene description can be used to support complete application specific interactivity. For example this can be used to support content selection by specifying the presentation in command parameters and the command ID indicates that the command is the content selection command. One way to implement this is to create an initial scene with several images and text that describes a presentation associated with an image. Associated with each image and the corresponding text is a content selection descriptor. When a user clicks on an image, the client transmits the command containing the selected presentation and server starts a new presentation. We have implemented the command descriptor support in our system. Initially only simple commands such as stream control commands and content selection commands are supported. More complex functionality such as electronic commerce and on line shopping shall be implemented in the next phase.

VII. CONCLUSION

In this paper we have described our MPEG-4 client server system with server interaction support. Delivering objet based audio-visual presentations is far more complex than traditional video on demand. We discussed several issues related to content creation, scheduling, delivery, and playback. MPEG-4 needs sophisticated content creation tools to create deliverable presentations. We also described the command descriptor framework that complements the VRML interaction model used by MPEG-4 to support server interaction in MPEG-4 systems. The implementation experiences are critical in developing the systems and improving the technology behind the MPEG-4 standards.

VIII. REFERENCES

[1] A. Akhtar, H. Kalva, and A. Eleftheriadis, "Command Node Implementation ", Contribution ISO-IEC JTC1/SC29/WG11 MGE99/4905, July 1999, Vancouver, Canada (48th MPEG meeting).

[2] O. Avaro, P. Chou, A. Eleftheriadis, C. Herpel, and C. Reader, "The MPEG-4 System and Description Languages: A Way Ahead in Audio Visual Information Representation", Signal Processing: Image Communication, Special Issue on MPEG-4, Vol. 9, No. 4, May 1997, pp. 385-431.

[3] M. C. Buchanan and P. T. Zellweger, “Scheduling Multimedia Documents Using Temporal Constraints,” NOSDAV 92, pp. 223-235.

[4] S.-F. Chang, A. Eleftheriadis, D. Anastassiou, S. Jacobs, H. Kalva, and J. Zamora, "Columbia's VoD and Multimedia Research Testbed with Heterogeneous Network Support", Journal on Multimedia Tools and Applications, Special Issue on Video on Demand, Kluwer Academic Publishers, Vol. 5, Nr. 2, September 1997, pp. 171-184.

[5] R. Civanlar, A. Basso, and C. Herpel, "RTP Payload Format for MPEG-4 Streams," draft-ietf-avt-rtp-mpeg4-01.txt, Internet Draft, IETF, February 1999.

[6] A. Eleftheriadis, “Architecting Video-on-Demand Systems: davic 1.0 and Beyond,” Proceedings, International Symposium on Multimedia Communications and Video Coding, Brooklyn, New York, October 1995.

[7] J.-F. Huard and A. A. Lazar, “A Programmable Transport Architecture with QOS Guarantees,” IEEE Communications Magazine, Vol. 36, No. 10, October 1998, pp. 54-62.

[8] J.-F. Huard, A. A. Lazar, K.-S. Lim and G. S. Tselikis, “Realizing the MPEG-4 Multimedia Delivery Framework,” IEEE Networks, Vol 12, No 6, 1998, pp 35-45..

[9] ISO/IEC/SC29/WG11, “Delivery Multimedia Integration Framework, DMIF (ISO/IEC 14496-6),” International Standards Organization, Feb. 1999.

[10] ISO/IEC/SC29/WG11, “Generic Coding of Moving Pictures and Associated Audio (ISO/IEC 14496-1),” International Standards Organization, May 1999.

[11] ISO/IEC/SC29/WG11, “Text of ISO/IEC 14496-1/FPDAM 1 (MPEG-4 Systems Version-2),” Doc. N2801, International Standards Organization, May 1999.

[12] ISO/IEC/SC29/WG11, "Text of ISO/IEC 13818-1/PDAM7," MPEG document N2664, March 1999.

[13] N. S. Jayant, “Signal compression: Technology targets and research directions,” IEEE Journal on Selected Areas in Communications, Special issue on speech and image coding, June 1992.

[14] H. Kalva and A. Eleftheriadis, “Use of CommandDescriptors and CommandROUTES to Support User Interaction ", Contribution ISO-IEC JTC1/SC29/WG11 MPEG98/M3506, March 1998, Tokyo, Japan (43rd MPEG meeting).

[15] H. Kalva, S.-F. Chang, and A. Eleftheriadis, "DAVIC and Interoperability Experiments", Journal on Multimedia Tools and Applications, Special Issue on Video on Demand, Kluwer Academic Publishers, Vol. 5, Nr. 2, September 1997, pp. 119-132.

[16] L. Kleinrock and A. Nilsson, “On Optimal Scheduling Algorithms for Time-Shared Systems,” Journal of the ACM, Vol. 28, No. 3, July 1981, pp. 477-486.

[17] M. Kunt, A. Ikonomopoulos, M. Kocher, “Second-generation image coding techniques,” IEEE Proceedings, Vol. 73, No. 4, pp. 549–574, April 1985.

[18] A. A. Lazar, “Programming Telecommunication Networks,” IEEE Network, pp. 8-18, September 1997.

[19] Z. Lifshitz, “APIs for System Software Implementation,” Contribution no. ISO/IEC JTC1/SC29 MPEG97/M3111.

[20] T.D.C. Little and A. Ghafoor, “Multimedia Synchronization Prootocols for Broadband Integrated Services,” IEEE Journal on Selected Areas in Communications, Vol 9, No. 9, Dec 1991, pp. 1368-1382.

[21] S. Paek and S.F. Chang, "Video Server Retrieval Scheduling and Resource Reservation for Variable bit Rate Scalable Video," To be published in IEEE Transactions on Circuits and Systems for Video Technology.

[22] L. Torres, M. Kunt, eds., “Video Coding : The Second Generation Approach,” Kluwer Academic Publishers, 1996.

[23] XDMIF demo software, http://www.xbind.com.

[24] J. Zamora, “Video-on-Demand Systems and Broadband Networks: Quality of Service Issues ,” PhD thesis, Columbia University, New York, NY, 1998.