Virtual Collaboration and Media Sharing using COSMOS

VASILIOS DARLAGIANNIS and NICOLAS D. GEORGANAS
Multimedia Communications Research Laboratory
School Of Information Technology and Engineering
University of Ottawa,
161 Louis Pasteur, Ottawa K1N 6N5, Ontario,
CANADA
[bdarla, georganas]@mcrlab.uottawa.ca http://www.mcrlab.uottawa.ca

Abstract: Collaborative Virtual Environments (CVE) are providing a powerful mechanism to companies, for training staff and customers in their products, improving their productivity, as well as reducing the cost of product development at the same time. Enhanced with their integration with media streams like audio and video, they can become the main way of collaboration among co-workers.

But, the realization of such a system is not a trivial task. Many issues like scene description, session management, stream sharing and synchronization are hard topics that require the development of a framework. COSMOS is such a framework that is based on the new multimedia standard, the MPEG-4 and the state-of-the-art development technologies, like Java 3D and JMF. In this paper, we describe the design and implementation issues of COSMOS and how it is integrated with collaborative applications.

Keywords: MPEG-4, BIFS, DMIF, VRML, RTP, Collaboration, Multicast, Virtual Reality

1 Introduction

COSMOS (COllaborative System based on MPEG-4 Objects and Streams) is an object-oriented framework that augments the rapid development of collaborative applications. Collaboration demands a set of requirements that have to be fulfilled in order to become acceptable. Some of those requirements are interactivity, short response times, robustness, and consistency. COSMOS offers the base for building new collaborative applications, with reusable, modular and extensible components, that fulfill the aforementioned requirements.

The MPEG-4 [6] [7] [8] [10] standard specifies a novel encoding technique, based on object description of audio-visual scenes. It specifies the scene description and its updates over time, the linking with media streams, and very efficient techniques to compress audio and video objects.

COSMOS has been implemented using the Java language. The Java Media Framework (JMF) is used for the encoding and decoding of the media streams (audio, video). It provides a set of encoders and decoders for the most popular encoding formats, as well as an implementation of the RTP protocol. The rendering of the 3D graphics is implemented using the Java 3D API. In addition, since VRML [9] is a very popular format, which is used to describe 3D worlds, a VRML parser has been developed using Java Compiler-Compiler (JavaCC), which is a Java-based parser generator tool.

In this paper, we present the architecture, network configuration, protocols, design and implementation issues of COSMOS. Our prototype contributes in the development of collaborative systems, by providing a framework implementation based on MPEG-4 technology. We describe how to develop MPEG-4 components with Java-based software technology. Also, we describe the structure of an example-application to show how COSMOS is integrated with other, collaboration-based applications.

The rest of this paper is organized as follows: Section 2 provides a short overview of MPEG-4. Section 3 describes the overall framework architecture and section 4 the session management issues. Section 5 provides the way a scene is represented in the system and section 6 the stream delivery issues. Section 7 gives some hints on the way applications should be developed over COSMOS and finally, section 8 summarizes the contribution of this paper.

2 MPEG-4 Overview

MPEG-4 is a new multimedia standard, much more powerful than the previous MPEG standards. It provides an object-based description of content, both naturally captured and computer generated. It specifies a set of video codecs for arbitrary shape of display to enable the extraction of visual objects from a scene. Also, it defines a set of audio codecs to cover every possible need of applications for trade-off between quality and compression rate.

MPEG-4 defines the Delivery Multimedia Integration Framework (DMIF) [2] [5 [7] [8]. DMIF is a framework that abstracts the delivery mechanisms from the applications. For example, multimedia applications require to access data from either a local hard disk, or from a network source with interaction, or even from a broadcast or multicast source. DMIF provides a unique API to the applications, through which they can request content or services. They don't have to "worry" where those data are. This API, called DMIF Application Interface (DAI), works with URLs, which specify the appropriate delivery mechanism. URLs can also specify the required network protocol, which provides a protocol abstraction for the applications. Quality of Service (QoS) requirements can be passed as arguments through this interface. DAI is language and platform independent. In addition, DMIF defines an informative DMIF-Network Interface (DNI) for the network related scenarios. DNI enables a convenient development of a framework that can easily change its signaling mapping to different protocols.

Moreover, MPEG-4 defines a way to describe the content of a scene with the Binary Format for Scene (BIFS). BIFS [1] [6] [14] [15] is based on VRML, which it extends to cover both 2D and 3D descriptions. In contrast to VRML - which is text-based - BIFS is providing a binary description of the nodes. Nodes are the basic entities, which construct a directed acyclic graph to represent the content of scene. Examples of nodes are Box, Transform, Shape or Group. BIFS enables the update of the content of a scene, as well as the animation of the objects with two protocols: BIFS-command and BIFS-anim. The information that BIFS provides is compressed and can be easily streamed among peers.

MPEG-4 is defined in two versions. MPEG-4 version 1 provides a basic set of tools and specifications, while MPEG-4 version 2 provides some more advanced and, hence, hard to be implemented features.

3 Framework Architecture

The general architecture of COSMOS is shown in Fig. 1 using a UML class diagram; it gives the big picture of the system. The focus of this figure is to show the way an application can access the functionality of each subsystem through the offered interfaces. It does not intend to provide all the details. Each subsystem is analyzed in the following sections.

The Application in this figure can have access to a configurable MPEG-4 Browser, that enables the navigation in a shared 3D world and the interaction with it. The MPEG-4 Browser communicates with an instance of the DMIFFilter. The DMIFFilter provides the DMIF Application Interface (DAI) to applications for the control of the establishment and release of the multicast sessions (DMIFSession). On the other hand, MPEG-4 Browser implements a DAI callback interface for the DMIFFilter (ApplicationSession). The DMIFFilter is responsible for handling all the requests for media access.

The MPEG-4 Browser encapsulates an instance of the SceneManagerImpl, which implements the SceneManager interface. The SceneManager interface provides a set of methods to manipulate the content of the scene, like adding or deleting a node, or modifying the value of a field. The SceneManagerImpl encapsulates a VRMLParser object that parses a VRML file and creates the corresponding directed acyclic graph. This graph is provided to the SceneManagerImpl. Similarly, the SceneManagerImpl has a BIFSEncoder to create streamable representations of the scenes and a BIFSDecoder object to generate scenes, which are encoded in BIFS format. The JMFCodec class is an abstract class that represents the set of encoders / decoders of JMF that encodes or decodes the multimedia streams.

The SessionManagerImpl implements the RTPSessionManager interface provided by JMF, to handle each media stream. It communicates with the DMIFFilter, which is making requests for new connections. The SessionManagerImpl encapsulates a set of RTPReceivers and RTPTransmitters to handle the details of receiving and transmitting RTP streams, respectively. The data from the RTP streams are provided to decoders, or retrieved from the encoders.

4 Session Management

The design of the DMIF subsystem that we implemented, is shown in Fig. 2, where the basic objects and interfaces, as well as their relationships are described using a UML class diagram.

Every MPEG-4 Application that uses the DMIF framework, must implement the ApplicationSession interface. It should also have access to a DMIFFilter instance, which provides the DAI by implementing the DMIFSession interface. The DMIFFilter informs the MPEG-4 Application for every change in the group state through the ApplicationSession callback interface.

DMIFFilter encapsulates the DMIF Instances that can handle specific DMIF scenarios. In the case of the multicast scenarios, those objects can be either GroupDCDTInstance or GroupDPDTInsta-nce, which represent a Data Consumer or a Data Producer, respectively.

Each of those objects encapsulates an IPMulticast object, which is responsible for the communication over the UDP/IP multicast protocol. IPMulticast implements the DNI interface, which is divided in two others, the DNIConsumer and DNIProducer. These interfaces define a set of methods for the respective type of DMIF terminal. IPMulticast uses a UDPSocket object to send and receive messages over UDP. Also, it requests from the TransactionManager to generate a transaction id and a Transaction object to handle the message exchange-based transactions.

GroupDPDTInstance and GroupDCDTInstance communicate with the RTPSessionManager to request the creation of specific transport channels.

5 Scene Representation

The inherited complexity of the scene management subsystem is handled via a well-defined object-oriented sub-framework, where a big number of objects are defined, with specific responsibilities. Fig. 3 shows the UML class diagram of the basic objects of this subsystem that we implemented.

The SceneManager class provides a set of methods, which enable the management of the content of the scene. SceneManager encapsulates three objects to handle scene content represented either in BIFS streams or VRML-like files. Two of them, the BIFSEncoder and BIFSDecoder encodes and decodes BIFS streams, respectively.

BIFSCodec encapsulates a static object, which is called NodeDataTypeInfo. NodeDataTypeInfo is the “gate” to a database, where information about the encoding of each node or field is stored (Node Data Types - NDTs and Node Coding Tables - NCTs). NodeDataTypeInfo implements the NodeProvider interface, to provide every node that is tagged with an id or structures that describes the way nodes and fields should be encoded/decoded. It encapsulates an object called NDTTable. It identifies the NDTs and chooses the right object. SFWorldNode is a special table, which keeps the actual info for all the nodes. The rest of the tables are referencing to this one. NodeData is a class that encapsulates the information, which is required to encode a node (an implementation of the NCT). Every node is related to such an object. For example, the TransformInfo object derives the NodeData object and stores all the information for the Transform node. Similarly, the FieldData object keeps information for every field.

BIFSCodec encapsulates one more important object, which is called BIFSScene. BIFSScene holds the directed acyclic graph, which describes the relationships of the nodes and a list, which includes all the active ROUTEs. In addition, it encapsulates the SFFieldTemplate objects. SFFieldTemplate is an abstract class, which is being derived from a set of objects responsible to encode fields of nodes. SFFieldTemplate is implementing the Streamable interface, which enables the output of the scene description into a binary stream, as well as the reverse procedure. A specific derived object, which is called SFNode is responsible to encode/decode nodes.

The actual representation of the directed acyclic graph is modeled with the Node objects and the corresponding Fields. Fields are deriving from the Observable object (observer pattern) to enable the sharing of the stored information to other interested fields. They also implement the Observer interface to be informed for changes in other fields. This way, the implementation of the ROUTEs is augmented.

In addition to the ability of encoding / decoding BIFS streams, the system can “read” VRML files. VRMLParser is responsible for that. It encapsulates a set of rules to parse valid VRML files.

6 Sharing of Media

As it has already been mentioned before, MPEG-4 content is described with a big number of elementary streams, which are composed at the MPEG-4 terminal to construct the rendering scene.

At the Delivery Layer, MPEG-4 does not define a specific transport protocol for the transmission of the elementary streams. Possible solutions are MPEG-2 TS, RTP, AAL2 ATM or H223. For transmission over the Internet, the most appropriate solution is the RTP [13] protocol or alternatively, with some restrictions, directly over UDP.

The Sync Layer provides a means for synchronizing the elementary streams, using timestamps and the appropriate buffering mechanisms. Data are forwarded for decoding at the correct time, and after they are composed and render at the MPEG-4 terminal. JMF and Java 3D are used for the decoding and the rendering of the media.

7 Implemented system

COSMOS has been developed using only the Java language. The framework has been tested under the Windows NT operating system, using Pentium processors with very powerful graphics cards. The rendering of the MPEG-4 content is handled by mainly two components that cooperate with each other. The 3D Renderer, which is a Java 3D implementation of an MPEG-4 node browser and the Video Renderer, which renders the decoded video frames onto the surfaces of 3D objects.

JMF provides a set of encoders/decoders to encode/decode video and audio streams, in many popular encoding formats, such as H.263, MPEG-1, AVI and Quick Time. The default rendering of the video stream is taking part into rectangular frames. It provides a variety of rendering implementations, such as 100% Java-based, or ActiveX-based. New rendering mechanisms have been implemented, by extending the functionality of JMF to render the video streams onto the surfaces of 3D objects.

The rendering of a 3D world, described with the Java 3D API is taking place into the Canvas3D object, which is a frame with 3D rendering capabilities. The Canvas3D object is connected to the Universe object, which is the top object in the hierarchy of a Java 3D implemented environment. An MPEG-4 node is a structure that keeps information regarding the values of the fields. It does not provide visual implementation by itself. For this reason, each MPEG-4 node is related to a Java 3D structure to enable the rendering of the information. This Java 3D structure can be either a Node or a NodeComponent. A Node can be either a Group node, such as BranchGroup or TransformGroup, or a Leaf Node, such as Shape3D or Light.

A representative example of a geometry node is the IndexFaceSet node. The IndexFaceSet node encapsulates the JMIndexFaceSet object, which is an implementation of the IndexFaceSet object, using Java 3D to render its geometry. It encapsulates a GeometryArray object (it is an object provided by Java3D API), to define the geometry of the node. In addition, it encapsulates information regarding the color of the node, as well as the normals and the some possible texture information. Similarly, other geometry nodes encapsulate the corresponding implementation nodes with Java 3D components.

8 Conclusions

In this paper, we presented COSMOS, a framework to support collaborative applications, based on the MPEG-4 standard. We show how to fill the gaps between the standard specifications and the prototype implementation. We show how to use the state of the art software development technologies, like Java 3D and JMF, to realize the MPEG-4 tools. We present a complete architecture for the description of the shared scene and the management of the sessions among the participants. Also, we describe how to deal with the elementary streams, according to their specific QoS requirements, to enable their multicasting. Finally, we give some hints on the way applications should be integrated with framework.

References:

[1] O. Avaro, A. Eleftheriadis, C. Herpel, G. Rajan, L. Ward, “MPEG-4 Systems: Overview”, Image Communication Journal, August 1999

[2] G. Franceschini, “The Delivery Layer in MPEG-4”, Image Communication Journal, August 1999

[3] C. Guillemot, S. Wesner, P. Christ, “Integrating MPEG-4 into the Internet”, ECMAST 99, 1999

[4] C. Herpel, A. Eleftheriadis, “MPEG-4 Systems: Elementary Stream Management”, Image Communication Journal, August 1999

[5] J. Huard, A. Lazar, K. Lim, G. Tselikis, “Realizing the MPEG-4 Multimedia Delivery Framework”, IEEE Network, July 1998

[6] ISO/IEC 14496-1 IS (MPEG-4), “Information Technology – Coding of audio-visual objects, Part 1: Systems”, http://flavor.ee.columbia.edu/docs/, January 1999

[7] ISO/IEC 14496-6 FCD (MPEG-4), “Information Technology – Coding of audio-visual objects, Part 6: DMIF”, http://drogo.cselt.it/mpeg/public/mpeg-4_fcd, May 1998

[8] ISO/IEC 14496-6v2 (MPEG-4), “Information Technology – Coding of audio-visual objects, Part 6: DMIF version 2”, 1999

[9] ISO/IEC 14772-1:1997, “Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML) -- Part 1: Functional specification and UTF-8 encoding”, 1997

[10] R. Koeman, “MPEG-4, Multimedia for our time”, IEEE Spectrum, http://drogo.cselt.it/mpeg/koenen/mpeg-4.htm, February 1999

[11] T. Liao, “Light-weight Reliable Multicast Protocol as an Extension to RTP”, 1997

[12] S. McCanne, “Scalable Multimedia Communication: Using IP Multicast and Lightweight Sessions”, IEEE Internet Computing, March/April 1999

[13] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson "RTP: A Transport Protocol for Real Time Applications" draft-ietf-avt-new-00, Internet Engineering Task Force, ftp://ftp.is.co.za/rfc/, Dec. 1998

[14] J. Signes, Y. Fisher, A. Eleftheriadis, “MPEG-4: Scene Representation and Interactivity”, Multimedia Systems, Standard, Networks, 1999

[15] J. Signès, Y. Fisher, A. Eleftheriadis, “MPEG-4’s Binary Format for Scene Description”, Image Communication Journal, August 1999

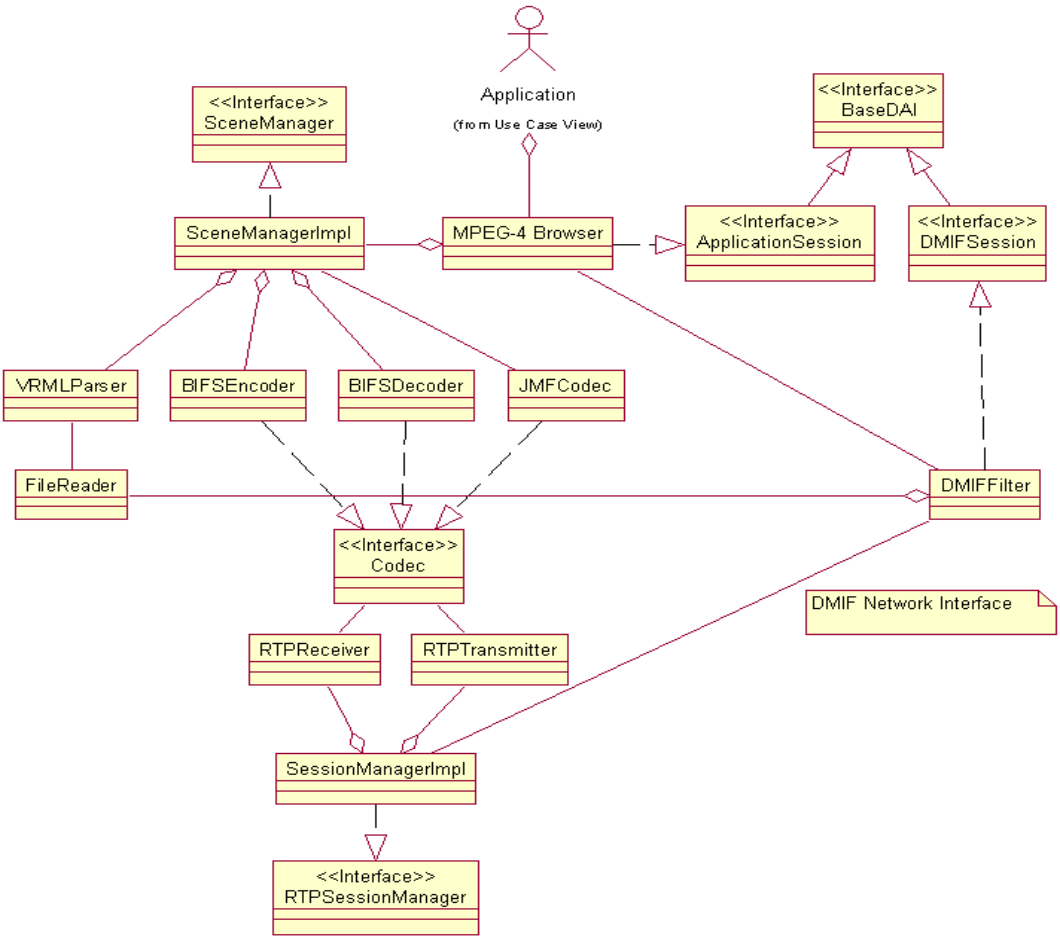

Fig. 1 Framework architecture

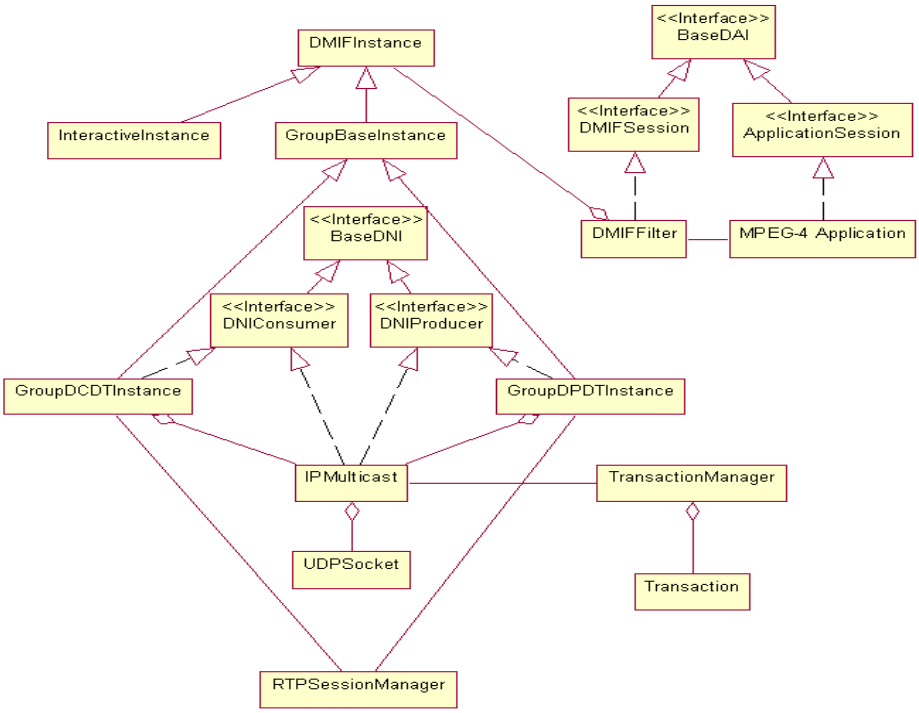

Fig. 2 DMIF architecture

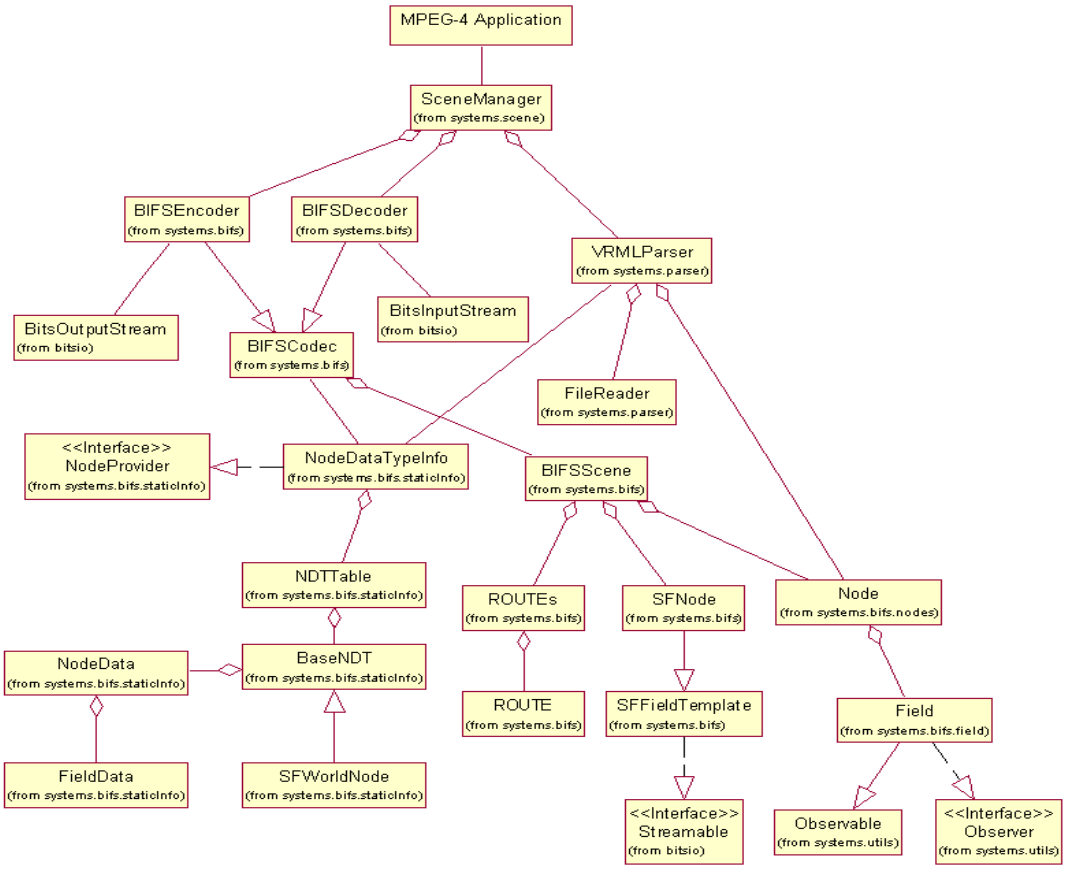

Fig. 3 Scene management architecture