INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa • University of Ottawa

COSMOS: COllaborative System framework based on MPEG-4 Objects and Streams

by

Vasilios Darlagiannis, Dipl. Ing.

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Applied Science
in
Electrical & Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Information Technology and Engineering
University of Ottawa

Abstract

Virtual collaborative systems enable cooperation of users, even when they are in different continents. Despite the fact that the best-effort nature of Internet hinders their operation over long distances, they can efficiently operate in relatively short distances. This is basically due to new compression standards that can efficiently adapt their Quality of Service (QoS) to the needs of the environment.

MPEG-4 is a novel International Standard that explores almost every possibility of the digital environment. It describes the content of a scene by identifying audio-visual objects into it, both of natural and synthetic nature. New compression techniques give very low bit-rate encoded streams for each identified object. Users can navigate and interact with the scene and share it with the other participants of the virtual environment. In this thesis, a new collaborative framework has been developed, based on the specifications of the MPEG-4 standard. The framework provides access to the content of the scene, which is constructed in a directed acyclic graph. The scene is compressed and streamed using the MPEG-4 Binary Format for Scene (BIFS) specifications. In addition, VRML files are supported and translated to BIFS. The rendering of the scene is based on Java 3D.

Media streams like audio and video are also rendered in the same scene with the 3D content. They can be rendered in the surface of every shape. The Java Media Framework (JMF) is used to decode and encode the audio-visual streams, and it is extended to enable their rendering in the 3D scene.

The transmission of the media streams among the participants of the shared environment is over the RTP protocol, using multicast delivery. JMF provides an implementation of the RTP protocol. This implementation is extended to provide a reliable multicast transmission protocol for the exchange of critical content, like the BIFS stream.

The developed framework provides an implementation of the multicast scenarios of the MPEG-4 Delivery Multimedia Integration Framework (DMIF). DMIF provides an interface to the applications with the required functionality to handle the establishment and the release of transport channels with QoS support.

Acknowledgements

I would like to thank Professor Nicholas D. Georganas for supervising me throughout the development of this thesis and for his trust and the opportunity that he gave me, by suggesting this interesting topic.

Also, I would like to acknowledge Professor Emil Petriu and Professor Dorina Petriu for the time they spent to review this thesis.

I would like also to thank my friends and colleagues in the MCRLab, and more especially Ramsey Hage, Shervin Shirmohammadi, Jauvane C. de Oliveira and Mojtaba Hosseini for their advice during the development of this thesis. In addition, I would like to acknowledge Francois Malric for his continuous administrative support.

Last, but not least, I would like to thank my family and my girlfriend for their endless love, patience and support, which helped me to complete this work.

Vasilios Darlagiannis

Ottawa, November 1999

List of Acronyms

3D	Three Dimensional
AAL	ATM Adaptation Layer
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AU	Access Unit
BIFS	Binary Format for Scene
BNF	Backus-Naur Form
Codec	Encoder/Decoder
DAI	DMIF Application Interface
DCDT	Data Consumer DMIF Terminal
DMIF	Delivery Multimedia Integration Framework
DNI	DMIF Network Interface
DPDT	Data Producer DMIF Terminal
DS	Default Signaling
DSM-CC	Digital Storage Media – Command and Control
FlexMux	Flexible Multiplex
GUI	Graphical User Interface
HMD	Head Mounted Display
IEC	International Engineering Consortium
IP	Internet Protocol
ISO	International Standardization Organization
JavaCC	Java Compiler Compiler
JDK	Java Development Kit
JMF	Java Media Framework
LAN	Local Area Network
LRMP	Light-weight Reliable Multicast Protocol
LWS	Light-weight Session
Mbone	Multicast Backbone
MIDI	

M PEG	Moving Picture Experts Group
MPEG-2 TS	MPEG-2 Transport Protocol
MPEG-J	Java Interface for MPEG-4
NCT	Node Coding table
NDT	Node Data Type
OCR	Object Clock Reference
OD	Object Descriptor
OMT	Object Modeling Technique
OOA&D	Object-Oriented Analysis and Design
OOSE	Object-Oriented Software Engineering
OTB	Object Time Base
QoS	Quality of Service
RLM	Receiver-driven Layered Multicast
RSVP	Resource ReSerVation Protocol
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol
RTSP	Real Time Streaming Protocol
SDM	Systems Decoder Model
SNHC	Synthetic Natural Hybrid Encoding
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
URL	Universal Resource Locator
VCR	Video Cassette Recorder
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
WAN	Wide Area Network
YACC	Yet Another Compiler Compiler

Table of contents

Chapter 1

1 Introduction

1.1 General

This thesis aims at the development of a collaborative system that supports sharing of multimedia streams and virtual environments at the same time. It is based on the specifications of the MPEG-4 standard, and provides a framework that offers an object-based description of scenes to every participant of the collaboration. Scene descriptions and multimedia are streamed to every terminal of the collaboration.

1.2 Multimedia

The word "multimedia" is composed of two words: Multi, which means many, and medium, which means a substance through which something is transmitted or carried on. As a computer science concept, multimedia has two components: A set of different media types and their integration.

Examples of media types are video or audio streams, still images, text, and graphics. Media types can be divided in two main categories:

- Time-independent media, where their information is not changed with time, like a text or a still image. They also called discrete media.
- Time-dependent media, where their information is changed with time, like a video or an audio stream. They also called continuous media.

Processing of continuous media types is based on strict time constraints, the violation of which leads to an abnormal presentation of the information. Moreover, the size of data of continuous media streams is related to their time length. Long media can have an extremely large size, which makes impractical their transmission over networks. For this

reason, many compression techniques have been developed, that minimize their size significantly.

A popular definition of multimedia systems is the following: "A multimedia system is characterized by computer-controlled, integrated production, manipulation, presentation, storage and communication of independent information, which is encoded at least through a continuous (time-dependent) and a discrete (time-independent) medium" [56].

1.3 Virtual Reality

The term Virtual Reality (VR) was first used by J. Lanier and J. J. Grinand. According to them, "Virtual Reality is a new plane of reality accessible through computerized clothing. A shared VR is one in which two or more participants are in a Virtual World at the same time, can see each other, and undertake joint projects" [46].

Although VR usually requires special equipment, such as Head-Mounted Displays (HMD) or data gloves, this system provides a simulation of a Virtual World, which is modeled using standard description techniques of virtual environments, and displayed in a computer screen. A less expensive VR system can be provided this way.

It should be noted that the terms Virtual Worlds and Virtual Environments are used alternatively to express a computerized model of VR.

1.4 Collaborative systems

Collaboration is the key to success in achieving important goals, due to the increasing complexity in almost every industry. But in many cases, the team members that have to cooperate are geographically separated, and hence, they need to travel long distances. Therefore, there is a pressing need for systems that would enable the tele-presence of co-workers in a virtual environment, without their physical co-presence in the same place. Moreover, the combination of multimedia streams into virtual environments provides a very powerful way to visualize and share information among the participants of the collaboration.

1.5 *Problem statement*

The aim of this thesis is to provide a framework that augments collaboration among users of interactive, virtual reality-based applications, integrated with natural-origin media streams.

This is a very challenging problem, which combines many different research areas. A collaborative system should provide a communication mechanism, which must be scalable to a big number of participants, without significant cost in the performance of the system. Every user can interact independently with the virtual world, navigate into it and modify its objects. Data consistency at each user's terminal should be provided, despite the distributed nature of the system.

Moreover, a mechanism to stream the content of the virtual world and compress it for more efficient transmission over networks is required. At the same time, an update mechanism should be provided. Media streams like audio and video improves the quality of the collaboration and gives much more powerful ways to display information. The rendering of media streams should be integrated with the 3D graphics of the virtual world to provide a more natural representation. Therefore, many different streams have to be exchanged among the participants; and each stream requires different Quality of Service (QoS) qualifiers. A session management mechanism should be provided to handle the establishment and release of each session, and creation of each data channel. Network usage should be very careful to enable the realization of such a system, where many users receive and send many streams. It requires an efficient delivery mechanism.

Many applications can take advantage of such a system, for example, e-commerce applications, tele-learning applications, tele-games or simulation applications.

1.6 *Proposed solutions*

The solution that this thesis is proposing is the development of a collaborative system based on the MPEG-4 standard.

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

Video objects do not have to follow the "traditional" rectangular shape, but they can have an arbitrary one. In addition, MPEG-4 specifies DMIF, a delivery framework to augment applications to access media streams from either broadcast sources, or storage devices, or remote services, or multicast groups, in a transparent way. It provides a session management mechanism with QoS support. Multicast groups are appropriate for a collaboration system, where a producer-participant multicasts its streams to many receivers concurrently.

The scene description is hierarchical and forms a directed acyclic graph. It is derived from VRML, but it is not text-based. It is described in a binary format (BIFS). Transmission of media over the Internet is accomplished over the RTP protocol, which provides real-time delivery of media, using the multicast mechanism of UDP. Moreover, in some cases of special streams, like scene description streams, the delivery of the data must be reliable. This system implements an extension of the RTP protocol to provide reliable transmission services.

For the encoding and decoding of the media streams (audio, video) JMF is used. It provides a set of encoders and decoders for the most popular encoding formats. The rendering of the 3D graphics is implemented using the Java 3D API. In addition, since VRML is a very popular format, which is used to describe 3D worlds, a VRML parser has been developed using JavaCC, which is a Java-based parser generator tool.

1.7 Contribution of the thesis

The developed system offers a configurable and extendible framework to applications that offer collaboration capabilities to the users. The design of the system is highly extendible and modular, so that each subsystem can be used separately. The following list provides the most important subsystems.

- Design and implementation of the MPEG-4 DMIF part, for multicast scenarios. This framework provides an API to applications to handle the establishment and release of multicast sessions. This DMIF implementation has also been used in a project, named "Quality of service monitoring and end-user control", which is funded by CITO and Nortel Networks and directed by Professor Bochmann. In addition, since the MPEG-4 Version 2 specifications are still a draft proposal, many suggestions have been made

to the DMIF development group regarding technical details of the standard, based on the experience earned from the development of this system.

- Design and implementation of a framework, which describes 3D scenes with a directed acyclic graph, based on the specifications of the MPEG-4 standard.
- Design and implementation of an MPEG-4 BIFS encoder/decoder tool, that is capable of encoding 3D scenes to MPEG-4 BIFS format and decoding BIFS streams to 3D scenes.
- Design and implementation of a rendering mechanism for 3D scenes, using the Java 3D API. In addition, a set of special renderers have been developed to render decoded video streams on the surfaces of 3D objects.
- Design and implementation of a VRML parser to "read" VRML files and construct a directed acyclic graph. It can be combined with the BIFS encoder to translate VRML text-based files to MPEG-4 BIFS streams.
- Design and implementation of a streaming framework to deliver the media to each participant of a collaborative group. JMF RTP API is used as a base for this part, and it is extended to support reliable multicast transmission. In addition, it is extended to support the transmission of MPEG-4 BIFS streams.
- Design and implementation of a subsystem for the interaction with the user, such as navigation into the virtual worlds, or modification of the parameters of the objects into it.
- Design and implementation of an application that demonstrates the usage of the basic functionality of the framework.

1.8 Structure of the thesis

The rest of the thesis is organized as follows:

Chapter 2 provides an overview of the required background information. Two ISO standards are described: MPEG-4 and VRML. In addition the RTP protocol is presented, followed by a short description of the UML language. Java is the implementation language of the system. Three extensions of the core language are described: JMF, Java3D and JavaCC.

Chapter 3 analyzes the design goals of the system and justifies its need. Chapter 4 provides the general architecture of the framework, described in UML.

The following 3 chapters are focused on specific subsystems of the framework. Chapter 5 describes the scene management and representation, which is based on the MPEG-4 BIFS format. It describes the subsystem architecture in UML, as well as the interaction of the most important objects. Chapter 6 describes the development of the session management subsystem, which is based on the specifications of the MPEG-4 DMIF part. Similarly to Chapter 5, the focus of this chapter is on the architecture and the interactions of the objects. Chapter 7 discusses the delivery and management of the elementary streams. It is focused on the way the system extends JMF RTP to provide the elementary streams to every participant of the collaboration.

Chapter 8 describes the basic requirements and the architecture of an application that uses the collaborative system.

The last chapter, chapter 9, summarizes the basic concepts of the thesis and discusses the most important of the possible future extensions of the system.

Three appendices give some extra information on the UML language, the way MPEG-4 BIFS encodes scenes and the architecture of the most important classes of the Java 3D API.

Chapter 2

2 Background Information

The purpose of this chapter is to provide the necessary background required by this thesis. In particular, the Virtual Reality Modeling Language (VRML), a language that enables the description of virtual 3D environments is briefly presented. MPEG-4, the standard that provides an object-based description of audio-visual scenes follows. The Real-time Transport Protocol (RTP), which is the protocol used for the transmission of media among the peers that participate in a virtual environment, is discussed. Moreover, Unified Modeling Language (UML) is briefly presented. UML provides a manner in which to describe software systems. Finally, two Java extensions, the Java Media Framework and the Java 3D API, which have been used for the development of this system are explained. In addition, Java Compiler Compiler (JavaCC) a Java Parser generator is discussed. JavaCC has been used for the development of the VRML parser.

2.1 Virtual Reality Modeling Language Overview

2.1.1 What is VRML

Virtual Reality Modeling Language (VRML) is neither virtual reality nor a modeling language. Virtual reality typically implies an immersive 3D experience (such as a head-mounted display) and 3D input devices (such as digital gloves). VRML neither requires nor precludes immersion. Furthermore, a true modeling language would contain much richer geometric modeling primitives and mechanisms. VRML provides a bare minimum of geometric modeling features and contains numerous features far beyond the scope of a modeling language [10].

VRML is a 3D text-based interchange format. It defines most of the commonly used semantics found in today's 3D applications such as hierarchical transformations,

light sources, viewpoints, geometry, animation, fog, material properties, and texture mapping.

VRML provides the technology that integrates three dimensions, two dimensions, text, and multimedia into a coherent model. The combination of these media types with scripting languages like JavaScript over the Internet provides an entirely new genre of interactive applications. VRML enables the visualization of their information and publication over the Web in a 3D way. VRML is the foundation for cyberspace and on-line virtual communities.

2.1.2 Design Goals and Constraints of VRML

VRML 1.0 specified static objects and scenes. Although it was very successful as a file format for the description of 3D objects and worlds, it did not support user interaction with objects and worlds, as well as animation of the objects. VRML 2.0 was created to extend the capabilities of VRML 1.0. VRML 2.0 includes user interaction and object animation. More advanced features, such as multi-user interaction or content streaming, are not a standard part of VRML 2.0.

The current version of the language, VRML 97, became an International Organization for Standardization (ISO) standard in December 1997. VRML 97 is similar to VRML 2.0.

2.1.3 Major Features

Scene data within a VRML file are organized into a hierarchical scene graph of nodes. VRML 97 defines 54 different node types, including geometry primitives, appearance properties, sound and sound properties, and various types of grouping nodes. Each node stores its data in fields. A field can be either a single- or multi-value field. VRML 97 defines 20 different types of fields that can be used to store data like, for example, a single number or an array of children nodes. The former example shows the way VRML builds a directed acyclic graph. Nodes can contain other nodes and may be contained in more than one node, but a node must not contain itself. This scene graph structure makes it easy to create large worlds or complicated objects from subparts.

In addition to the directed acyclic graph, a VRML file may contain a list of ROUTE statements. ROUTE statements define event paths between event generators and receivers. VRML 97 defines an event or message-passing mechanism by which nodes in the scene graph can communicate with each other. User interaction and object animation is achieved mainly through a set of nodes, called sensors. For example, the TimeSensor node generates events as time passes and is the basis for all animated behaviors. Sensors generate events, which have to be connected with the right nodes by defining a ROUTE statement. More advanced visual effects can be achieved using script nodes. Scripts allow users to define complicated behaviors in Java or JavaScript language. Interpolator nodes are built-in scripts that perform simple animation calculations. They are usually combined with a TimeSensor and some node in the scene graph to make objects move.

VRML allows reusing of nodes with the DEF and USE statements. The DEF statement defines a new node and gives it a reference name and the USE statement is referring to this node using that name. The PROTO statement allows the definition of a new node, combining existing node types.

VRML allows distributed scenes with the Inline node where the URL of a new scene is defined in one of its fields. The EXTERNPROTO statement allows nodes to be defined in other VRML files and enables their fetching from the Internet.

For a more extended reading about VRML the reader can refer to the specifications of the standard [32], [33].

2.2 MPEG-4 Overview

2.2.1 General

MPEG-4 is the latest international standard for multimedia applications. Its goal is to provide a new kind of standardization that responds to the evolution of multimedia technology, while it can be configurable for many multimedia applications by allowing flexibility in the systems configuration. MPEG-4 became an International Standard on January 1999, and it was called MPEG-4 Version 1. This version provides a basic functionality for highly interactive multimedia applications in an object-oriented way. An extension of this version is under development and it is expected to reach the status of

International Standard at the beginning of 2000. The new extended specifications compose the MPEG-4 Version 2.

2.2.2 *Scope and Focus of the MPEG-4 Standard*

MPEG-4, as referenced in [49], was originally intended for very high compression coding of audio-visual information at very low bit-rates of 64 kbit/s or under. When MPEG-4 video encoding techniques were investigated, it was expected that a scheme capable of achieving very high compression would emerge for standardization. By mid 1994, it was realized that the new video coding schemes were offering only moderate increase in compression (say, by factor of 1.5 or so), compared to the already existing techniques. In addition, a new class of multimedia applications was emerging that required increasing levels of functionality than those provided by any other video standard at bit-rates in the range of 10 kbit/s to 1024 kbit/s.

This lead to broadening of the original scope of MPEG-4 to larger range of bit-rates and important new functionality. MPEG-4 was aimed at providing new audio-visual coding solutions to allow interactivity, universal accessibility and sufficient compression.

2.2.3 *Basic Features*

MPEG-4 is a multimedia standard, rich in advanced features, which explores almost every possibility in a digital environment. It combines 3D with 2D features and naturally captured with computer generated audio-visual objects, in a highly interactive system. User's interaction is not limited in the traditional VCR functionality, like start or stop a video stream. Users interact with objects within the scene, navigate into 3D worlds, and even collaborate with other users. They have the power to modify the scene, by adding, deleting or repositioning objects.

Also, since recent digital copying of audio-visual streams has become a popular practice, MPEG-4 is designed with features for protection of intellectual property and digital content.

MPEG-4, as has already been mentioned, deals with objects, either visual or aural. Objects are grouped at the MPEG-4 terminal to compose an MPEG-4 scene. Objects have a position in a 3D or 2D space. The description of the scene is defined using a binary

stream, which is specific for scene description, the Binary Format for Scene (BIFS). BIFS borrows many concepts from VRML. Their main difference is that VRML is text-based, while BIFS has a binary format. This gives BIFS the advantage of being shorter for the same content, typically 10 to 15 times. In addition, BIFS can dynamically update the content of a scene, and can be streamed to the MPEG-4 terminal.

MPEG-4 provides a way to multiplex streams, as well as to synchronize their presentation at the terminal. Multiplexing is achieved using the FlexMux tool, a tool that can multiplex streams, if the underlying transport protocol does not support multiplexing, or because the overhead is very expensive. Synchronization is achieved at the Synchronization Layer, where elementary streams are packetized and time-stamped to enable their synchronized composition and presentation. MPEG-4 does not define a specific transport protocol. Existed transport formats can be used, like Real-time Transport Protocol (RTP) or MPEG-2 Transport Stream [16] [27]. The first is more appropriate for transmission over the Internet, while the former is more suitable for digital broadcasting.

MPEG-4 provides also a delivery framework, called Delivery Multimedia Integration Framework (DMIF). DMIF provides three types of abstraction. The first is the abstraction of the application from the delivery type. Interactive (client-server), local, broadcast or multicast delivery is controlled and accessed through a unique API. The choice of the appropriate delivery mechanism is based on the parsing of the URL that locates the required data or service. Quality of Service (QoS) parameters are also passed to determine the requirements of the application. The second is abstraction from the transport protocol, which can be RTP/UDP/IP or AAL5/ATM or MPEG-2 TS or others. Further, in the interactive or multicast case, DMIF provides an abstraction from the signaling mechanisms of the delivery system.

MPEG-4 video encoding supports the traditional rectangular video frames, but also, objects with arbitrary shapes can be encoded apart from their background and then composed with other streams in an entirely new scene. Of course, MPEG-4 does not specify how shapes are to be extracted from a scene. MPEG-4 video encoding provides many advanced techniques to ensure robust transmission over networks with high probability of error, like mobile networks. For example, resynchronization points are

introduced, to enable the recovery after those points. Also, reversible variable-length code techniques can be used. These techniques allow unique decoding even when the encoded data are read backwards, providing the advantage of decoding all the uncorrupted data.

Furthermore, MPEG-4 video encoding can be achieved in a scalable way. A base layer conveys all the information in some basic quality, and one or more enhancement layers can provide a better quality, if the transmission bandwidth is available. This technique allows unequal error protection. The most important streams – usually the base layer – can have more protection information.

The MPEG-4 standard provides several tools for audio encoding., starting at 6 kbits/s and reaching beyond 128 kbits/s, for natural audio. Speech is also handled with two algorithms. One is parameter encoding which can operate at a bit-rate of 2-4 kbits/s, or even lower in variable bit-rate encoding. Aural objects can be mixed, synthesized and rendered in a 3D virtual environment [37].

As has already mentioned, MPEG-4 Version 2 is under development. The new version, will include more advanced and complicated parameters, like advanced BIFS stream, multicast scenarios for the DMIF part, a Java interface for MPEG-4 systems, called MPEG-J, etc.

It should be noted that this section does not provide all the features of MPEG-4, but the most important. For more information the reader can refer to the specifications of the standard [29] [30] [31].

2.2.4 *Organization of the MPEG-4 Standard*

The MPEG-4 standard (ISO/IEC 14496) consists of the following basic parts:

- ISO/IEC 14496-1: Systems, which standardizes the development of techniques for multiplexing/de-multiplexing, composition and presentation of moving images, audio, audio, graphics and data [29].
- ISO/IEC 14496-2: Visual (Natural and Synthetic Video), which standardizes the development of coded representation of moving pictures of natural as well as synthetic origin, by cooperating with the Synthetic and Natural Hybrid Coding (SNHC).

- ISO/IEC 14496-3: Audio (Natural and Synthetic Audio), which standardizes the development of coded representation of audio of natural as well as synthetic origin, by cooperating with the SNHC.
- ISO/IEC 14496-4: Conformance, which provides methods for subjective and objective assessment and conducts tests.
- ISO/IEC 14496-5: Software, which evaluates the realization of the coding techniques.
- ISO/IEC 14496-6: DMIF, which standardizes the interfaces between digital storage media, networks, servers and clients for delivery bit-streams in networked environments [30] [31].

2.3 Real-time Transport Protocol Overview

The Real-time Transport Protocol (RTP) is a transport protocol, which represents a new style of protocol following the principles of application level framing and integrated layer processing. It provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. Those services include payload type identification, addition of sequence numbers and time-stamps and also delivery monitoring. RTP is usually used on top of UDP to make use of its multiplexing and checksum services. However, RTP can be used with other suitable underlying network or transport protocols. RTP can be used over both unicast and multicast network services.

RTP itself does not provide any mechanism to ensure maximum delay or delay variation or provide other Quality-of-Service (QoS) guarantees, but relies on lower-layer services to do so. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence. The header of each RTP packet is shown in Figure 2.1.

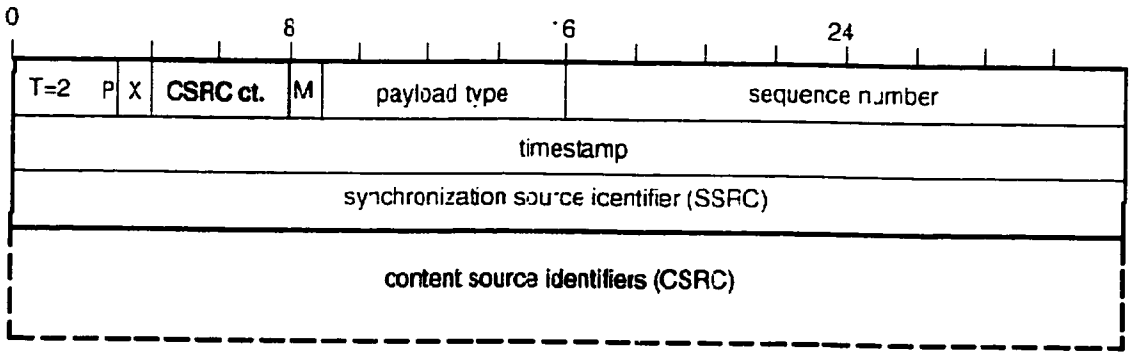

Figure 2.1 RTP packet header

RTP is consisted of two closely linked parts:

- The Real-time Transport Protocol (RTP), to carry data that have real-time properties.
- The RTP Control Protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session.

RTP is a general transport protocol that can be used for transmission of different media types and encoding. The way that a specific payload type should be carried in RTP is defined in payload format specification documents. Also, a profile specification document is required to define a set of payload type codes and their mapping to payload formats [52].

RTCP supports the protocol functionality. An RTCP message consists of a number of packets, each with its own type code and length indication, which can be compound to larger packets. Their format is fairly similar to data packets. RTCP packets are multicast periodically to the same multicast group as data packets. RTCP packets contain the necessary information for QoS monitoring.

2.4 Unified Modeling Language Overview

2.4.1 What is UML

Unified Modeling Language (UML) [8] is a modeling language for object-oriented analysis and design (OOA&D). UML unifies three older important methodologies for OOA&D:

- Booch notation, by Grady Booch
- Object Modeling Technique (OMT), by Jim Rumbaugh and

- Object-Oriented Software Engineering (OOSE), by Ivar Jacobson

UML is a modeling language that contains a graphical notation, which describes the syntax of the language. UML also has a meta-model that defines formally the language semantics.

UML is used to specify, construct and document software systems at all levels of the development.

It is also important to be mentioned that UML is accepted by OMG and it is supported by industry's leading software producers, like IBM, Microsoft, HP, ObjectTime, Oracle, Unisus, etc.

2.4.2 UML Basics

UML provides five views to describe a system:

- Use-case view, which describes the functionality of a system, as perceived by external actors that interact with it.
- Logical view, which describes how the system functionality is provided inside the system.
- Component view, which describes the implementation modules and their dependencies.
- Concurrency view, which deals with the division of the system into processes and threads, the allocation to different processors and the parallel execution of them.
- Deployment view, which shows the physical deployment of the system and their interconnections.

Each view contains a number of diagrams describing a certain aspect of the system. Diagrams are graphs describing the contents in a view. UML has several types of diagrams that are used in combination to provide all views of the system. Some of the most important diagrams are:

- Use-case diagram, which describes interactions between users (external agents) and the system.
- Class diagram, which describes the static structure of the classes and their relationships.
- State diagram, which describes the class behavior.

- Sequence diagram, which describes sequences of messages exchanged between objects represented with timelines.
- Collaboration diagram, which describes messages between objects represented on an object diagram.

UML has been used to describe the architecture of this system. More details with some examples are given in Appendix A.

2.5 *Java Platform*

Java is a relatively new object-oriented programming language. Java has a standard body that developers are using to create their systems and applications, the Java Development Kit (JDK). There is a big literature for the JDK, and, since Java is very popular, it will not be presented in this document. But, Java has also many extensions, which are specialized APIs and libraries. Three of those extensions have been used for the development of this system. The Java Media Framework, the Java 3D API and the JavaCC tool. They are briefly discussed in the next three subsections.

2.5.1 *Java Media Framework*

2.5.1.1 General

The Java Media Framework (JMF) API enables the display, capture, encoding, decoding and streaming of multimedia data in Java technology-based applications and applets. As a standard extension to the Java platform, it delivers the ease-of-development and cross-platform benefits of the Java programming environment to developers incorporating media data such as audio and video into their applets and applications. The JMF API specifies a unified architecture, messaging protocol and programming interface for playback, capture and conferencing of compressed streaming and stored timed-media including audio, video, and MIDI across all Java compatible platforms [58]. A part of this extension provides an API for the RTP transport protocol, which is appropriate for transmission of real time media.

The current implementation of JMF is a beta version that supports media capturing and presentation, as well as encoding and decoding of them. It is provided by SUN Microsystems and IBM.

2.5.1.2 Architecture

JMF is an extendible framework that integrates every possible encoding technique for audio and video. It can also be extended to include different media types, although this is more complicated.

Figure 2.2 shows the general architecture of the most basic components of JMF. The application can access the methods of those objects through the JMF API. A Data Source encapsulates the media stream, which can be accessed through a URL. The Data Source provides this media stream to either a Player or to a Processor. A Player is an object that renders the media stream, either to a screen or to speakers. A Processor is an object that can manipulate the media stream and transcode it to a different format or to process the data and add some special effects. It also de-multiplexes the media stream, if this is a multiplexed one (e.g. MPEG-2 Systems, which might contain an audio and a video stream). The Processor can provide its output stream to a Data Sink object that can either feed a network connection or save it to a local file. A Processor can also provide its output to another Data Source and then to another Processor object for further processing. A Processor or a Player can be constructed from the Manager object, which implements the Factory pattern.

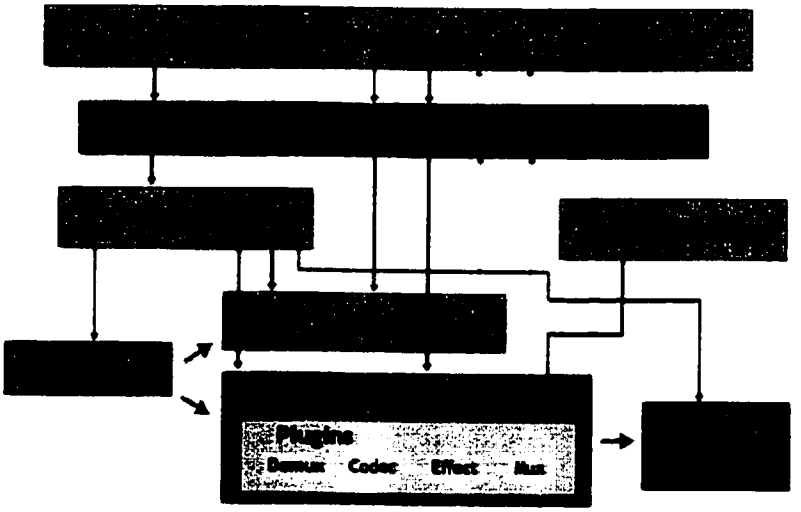

Figure 2.2 JMF architecture [43]

2.5.1.3 Extension mechanism

Figure 2.2 also shows one more object, the Plug-In Manager. The Plug-In Manager is a very important object, which allows the extension of JMF. It enables extension in two different ways:

- Adding a new service, such as a new protocol (e.g. DMIF)
- Adding a new encoding technique (e.g. MPEG-4 Video).

The Plug-In Manager can add new codecs, multiplexers/demupliplexers, packetizers/depacketizers, renderers or effects.

In addition, the user should support the framework with the appropriate Data Sources, Data Sinks and media Handlers.

2.5.1.4 JMF RTP

As it has already been mentioned, JMF provides a RTP API to enable the transmission of real-time streams. The user can access the RTP functionality through a Session Manager object, which provides an interface to start a new session at a specific address and port. The application can listen for new streams and create new Data Sources and Players or Processors or Data Sinks to manipulate them. JMF RTP allows a more low level access to RTP specific objects, where the user can create its own sessions with specific characteristics and even RTP extensions, such as an upper level which can provide reliable transmission services. Figure 2.3 shows the basic architecture.

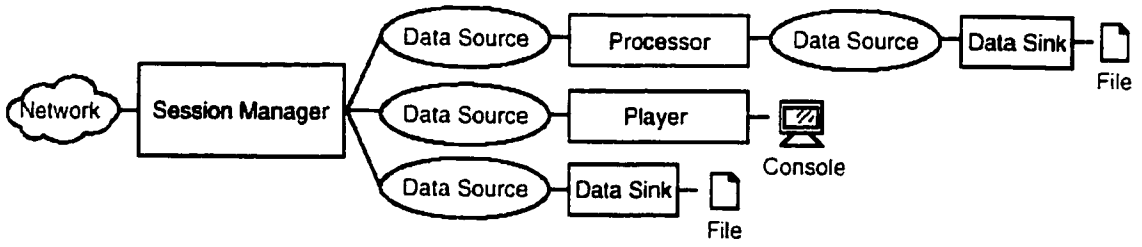

Figure 2.3 JMF RTP architecture [57]

2.5.2 *Java 3D*

2.5.2.1 General

The Java 3D API is an application programming interface used for writing stand-alone 3D graphics applications or Web-based 3D applets. It gives developers high level constructs for creating and manipulating 3D geometry and tools for constructing the structures used in rendering that geometry. In addition, it offers basic functionality to developers to define objects behavior, animation, as well as user interaction with the scene. With Java 3D API constructs, application developers can describe very large virtual worlds, which, in turn, are efficiently rendered by the Java 3D API [60].

Currently there are two different implementations provided by SUN Microsystems and its partners. One that is based on Open GL and one that is based in Microsoft's DirectX.

2.5.2.2 Architecture

Java 3D is an object-oriented API. Applications construct individual graphics elements as separate objects and connect them together into a treelike structure, which is called a scene graph. Figure 2.4 shows the scene graph for a simple application. The same model is followed in more complex cases.

Every scene graph starts with a VirtualUniverse object, which defines a three-dimensional space with an associated set of objects. Every VirtualUniverse object includes a list of Locale objects[1]. The Locale object acts as a container for a collection of subgraphs of the scene graph. A BranchGroup object is the root of every subgraph. A Locale also defines a location within the virtual universe using high-resolution coordinates[2] to specify its position. The high-resolution coordinates serve as the origin for all scene graph objects contained within the Locale.

One specific subgraph serves as a viewing subgraph. In this subgraph, a ViewPlatform object is attached, which controls the position, orientation and scale of the viewer. It is

---

[1] In this figure there is only one Locale

[2] High resolution coordinates are 256 bits long

the node in the scene graph that a View object connects to. The View object specifies information needed to render the scene graph. Also, an avatar, which is a 3D representation of the user in the virtual environment, can be attached in the View object. In this way, collaborative environments can be created.

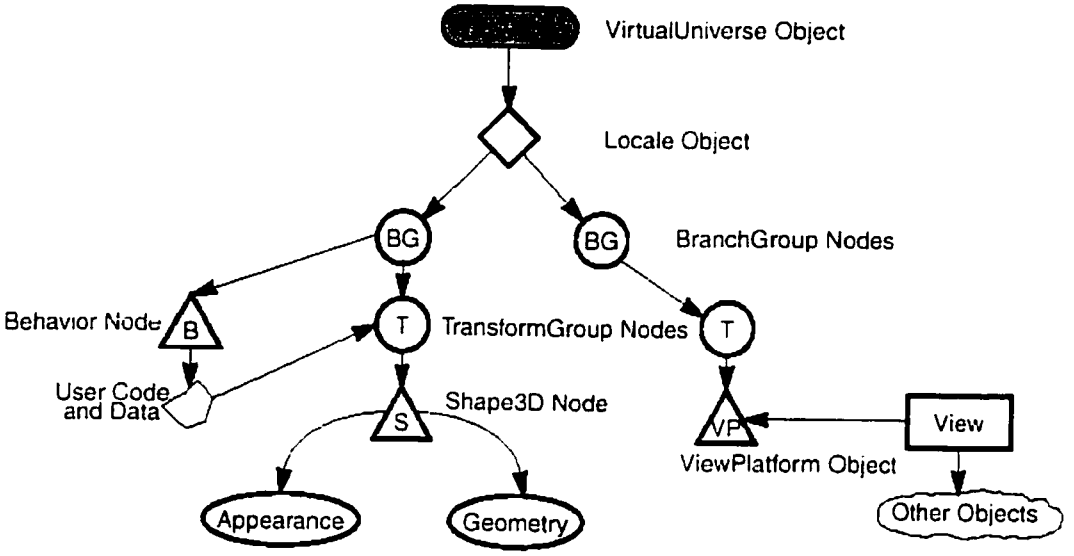

Figure 2.4 Application scene graph [60]

The nodes in a branch subgraph fall into one of two categories:

- Group nodes, which combine other nodes into one common unit such as branches and transforms.
- Leaf nodes, which represents items in a branch subgraph such as shapes, lights and behaviors.

Leaf nodes do not refer to other nodes, but only to other Java3D objects, like NodeComponent objects. These objects hold Java 3D specific information such as shape geometry, appearance, textures, media containers, and so on [60]. Appendix C gives some additional information about the hierarchical structures of the classes of the Java 3D API.

2.5.3 Java Compiler Compiler

Java Compiler Compiler (JavaCC) [64] is a Java parser generator written in Java. It takes a language described in a special format as input and it produces pure Java code.

JavaCC generates top-down (recursive descent) parsers as opposed to bottom-up parsers generated by YACC[3]-like tools. This allows the use of more general grammars (although left-recursion is disallowed). Top-down parsers have a set of other advantages (besides more general grammars) such as being easier to debug, the ability to parse to any non-terminal in the grammar, and the ability to pass values (attributes) both up and down the parse tree during parsing.

The lexical specifications such as regular expressions, strings, etc. and the grammar specifications (the BNF) are both written together in the same file. It makes grammars easier to read (since it is possible to use regular expressions inline in the grammar specification) and also easier to maintain.

JavaCC comes with JJTree, a tree building preprocessor. JavaCC offers many different options to customize its behavior and the behavior of the generated parsers. Examples of such options are the kinds of Unicode processing to perform on the input stream, the number of tokens of ambiguity checking to perform, just to name a few.

By default, JavaCC generates an LL(1) parser. However, there may be portions of the grammar that are not LL(1). JavaCC offers the capabilities of syntactic and semantic lookahead to resolve shift-shift ambiguities locally at these points. i.e., The parser is LL(k) only at such points, but remains LL(1) everywhere else for better performance. Shift-reduce and reduce-reduce conflicts are not an issue for top-down parsers.

JavaCC allows extended BNF specifications - such as (A)*, (A)+ - within the lexical and the grammar specifications. Extended BNF relieves the need for left-recursion to some extend.

JavaCC offers "lex"[4]-like lexical state and lexical action capabilities. Specific aspects in JavaCC that improve over other tools are the first class status it offers concepts such as TOKEN, MORE, SKIP, state changes, etc.

JavaCC has been used to develop the VRML parser.

---

[3] Yet Another Compiler Compiler (YACC). A similar tool, that produces C code.

[4] A popular token analyzer, which works with YACC

Chapter 3

3 System Design Goals

The main goal of this work is to provide a system based on international standards and not on proprietary solutions. Compatibility with international standards increases its interoperability with other systems and applications provided from other vendors, and enables their easy integration. Also, like every real-time system, performance is a key-characteristic for its success. Although this system is a prototype, a special attention has been paid on the time-consuming processes like coding/decoding and rendering of media streams. Furthermore, the stability of memory usage of the whole system in general, is another important point.

Despite of the general-purpose goals, the system has to fulfill some subject-specific ones. Those goals are analyzed in the following subsections.

3.1 Collaboration and Interaction

Collaboration is the key to success in achieving big goals, due to the increasingly complexity in almost every industry. But in many cases, the team members that have to cooperate are geographically separated, and hence, they need to travel long distances. Therefore, there is a pressing need for systems that would enable the tele-presence of co-workers in a virtual environment, without their physical co-presence in the same place. Although it is not still possible to develop such a system, mainly because of lack in network support, it is predicted that it will be a reality in the near future. The basic problem is the best-effort nature of Internet, where there is no guaranty for a specific required Quality of Service (QoS).

Despite this weakness in network support, there is a big research effort in collaboration systems. Such systems can be currently developed to operate, for example, over Local Area Networks (LANs).

Many recent advances in Internet technologies, as well as in multimedia standards, add diversity and new dimensions to computer-supported collaboration. Some of them are:

- The ability for IP multicasting,
- The improvement in performance of platform-independent languages, like Java,
- The advances in multimedia streaming and the development of systems for computer generated graphics,
- The advances in multi-agent systems.

In this thesis, the developed system is aiming in providing a framework for collaboration in a highly interactive multi-user environment. It aims to support the integration of natural video and audio streams with computer generated ones, as well as the synchronized rendering at each terminal. It should be pointed that this system is not aiming at solving every problem regarding the collaboration of co-workers, but at providing a framework that will be the basis on which a more complete system could be developed.

More specifically, the aim of this system is to provide a highly scalable collaboration framework, which can work efficiently with many users, without their large number to significantly decrease the overall performance. Also, "late-comers", meaning users that join the collaborative environment after some collaboration has taken part, are informed of those changes. Every participant can configure the parameters of its system according to the available system resources. For example, he can decide to accept only computer-generated streams, which in general require less bandwidth than their natural counterparts.

3.2 Streaming over the Internet

Streaming over the Internet, even if the multimedia streams are encoded at very low bit rate, is a very challenging problem, mainly due to the fact that Internet is a "best-effort" network. This means that it offers no guarantees for QoS requirements that are necessary for acceptable operation of the applications. Problems are coming from buffer overflows in the network nodes in cases of congestion. Based on the current architecture of the Internet, congestion can not be avoided. Internet sources start transmitting data

independently from each other, to arbitrary destinations, without reserving the required bandwidth (RSVP might be a partial solution). But, even if congestion can not be avoided, actions should be taken, first to detect the congestion and second to help the network overcome the temporal situation. In addition, error protection mechanisms can be implemented to detect and correct errors due to transmissions. Priorities can be given to handle differently the most important data (i.e. data, the loss of which will have bigger influence in the QoS).

In the following subsections, some of the basic features of multimedia streaming over the Internet are discussed together with the goals of this system.

3.2.1 Multicasting

IP Multicast extends the traditional best-effort unicast delivery model of the Internet Protocol with efficient multipoint packet transmission, thereby enabling a broadcast style of communication that lets large collections of Internet hosts exchange network packets efficiently [42]. Multicast is different than broadcast since only "subscribed" members receive the packets. Thus, multicast makes a much better use of the network resources than broadcast and it is much more efficient than unicast.

Members of a multicast group are identified by a single group address rather than by their individual host address. The basic characteristics of the multicast architecture, as they are referred in [39], are described in the following list:

- The Internet supports group addressing of multicast packets. This addressing eliminates the need for the source to know the identity of all the receivers, which is appropriate for a scalable multicast infrastructure.
- Packets are delivered to receivers that have declared their membership in the group over a tree that spans all such receivers.
- When a new receiver joins the multicast group, a path reaching this new receiver is grafted onto the tree if necessary. When a receiver leaves a group, any paths within the tree that are no longer needed are pruned.

Although most commercial Internet devices support IP Multicast, there are not sufficient routing protocols, for a global use. Fore this reason, a virtual multicast network

is overlaid on top of the Internet, which is called Multicast Backbone (MBone). Of course, multicasting can be easily implemented in a Local Area Network (LAN).

Because of its important advantages, multicasting has been chosen as the delivery model for the communication among the users that collaborate in the virtual environments. Of course, there are many challenges that have to be faced, like reliable transmission for specific streams, or management of the multicast sessions. Those issues are discussed in the following subsections.

3.2.2 Real-time Transmission

Digital audio and video are time-continuous media. They must be sampled and played at regular intervals to provide an acceptable perceived quality. This implies that, in a distributed multimedia system, the audio-visual streams should be transmitted under specific time constraints, regarding delay, delay variation (jitter) and packet losses. In addition, for their transmission over the Internet, they should be "fair" with respect to other streams. This means, that in case of congestion, the transmission rate should be decreased, similar to the behavior of the Transmission Control Protocol (TCP).

Jitter can be faced with the addition of a buffer at the receiver, which reduces the variations at the arrival times. The size of that buffer should be reasonable, especially in case of interactive applications. A big buffer eliminates the problem of jitter, but it adds a big end-to-end delay, which can be unacceptable in some cases.

The transmission of time-continuous media does not need to be reliable. This means that there is no requirement for a guaranty that every packet will arrive at the receiver. Of course, a big packet loss reduces the perceived quality. There are many techniques that are trying to solve this problem. In the case of multicast transmission, one of the most popular is the Receiver-driven Layered Multicast (RLM). In RLM, a video stream is encoded using a stream for basic quality and one or more enhancement streams to improve the quality. The sender sends each video layer to a separate IP multicast group and takes no active role in rate adaptation. Each receiver subscribes to a certain set of video layers by joining the corresponding IP multicast group. The advantage of receiver-based control over sender-based control is that the burden of adaptation is moved from the sender to the receivers, resulting in enhanced system scalability.

A different solution is the sender-based control. The sender receives feedback from the receivers regarding the number of lost packets and adjusts accordingly the transmission rate. The advantage of this solution is that it can be used when there is only one encoding layer available and the previous solution is not feasible; it is the case that this system is adapting. The sender is decreasing the number of frames per second of a stream, when a predefined portion of receivers reports big packet loss. The frames that are chosen to be dropped, are depending on the encoding scheme. For example, if the video stream is MPEG-1 video, then the sender can drop the B-frames to reduce the required bandwidth.

3.2.3 Reliable Transmission

Except the time-continuous media that are exchanged among the participants of a collaboration system, there are also non-time-continuous streams. An example is this type of streams is the MPEG-4 BIFS stream. This stream is describing the virtual scene of the collaborative environment. It should be reliably received by every participant, to have the correct representation of the scene and of the possible actions of the other participants. It contains all the interactions of the users with the scene objects.

In this transmission service, the sender should retransmit every lost packet, even when only one participant didn't received it correctly.

A more advanced technique can be a protocol that is giving priorities to the actions, and permits unreliable transmission of non-important information. For example, loss of an intermediate position of an object might not be a problem, as long as every participant reliably receives the final position. This kind of protocols is under research.

3.2.4 Quality of Service

The Quality of Service (QoS) in real-time systems is a very challenging issue. Regarding the transmission QoS, as it has already been mentioned, a server-based control is adjusting the transmission control to maximize the overall QoS. Also, every receiver has a buffer to reduce the transmission jitter.

In addition, there is a need for every application that uses this framework to be able to request for a specific QoS, and to be informed when the requested QoS can not be met.

MPEG-4 DMIF provides that interface for every application. It is in charge of the application to take the right actions to handle a situation where the requested QoS can not be met.

3.3 Session Management

The network delivery model for the communication among the participants of a virtual world is based on group communication mechanisms. The session management for this model should be scalable, as well as "lightweight". In literature, this type of session management is called Lightweight Sessions (LWS) [42]. It is a loosely coupled and decentralized communication model that is based on IP Multicast.

MPEG-4 DMIF defines an appropriate protocol that is robust to network failures. It is a "soft state" protocol that is based on an "announce/listen" model, in which each session member periodically announces its presence by multicasting a message to the group. Receivers are informed of the available services and channels and can connect to those that they are interested in. The state of each sender is updated by new announcements. Every announcement can be valid only for a specific amount of time.

The big advantage of this approach is the similarity of the delivery models between the session management and the data transmission channels. They are both based on group communication mechanisms. This makes much easier the description of the data channels in the session messages.

3.4 Stream Synchronization and Multiplexing

There are mainly two different synchronization aspects for multimedia streams:

- Inter-stream synchronization, which refers to the temporal relations linking the various media streams within a multimedia presentation.
- Intra-stream synchronization, which refers to the temporal relations between media units (e.g. frames), within a time-dependent media stream.

Synchronization can be handled in each terminal using the available time-stamps in each RTP packet. That time-stamp contains information on when each packet should be available in the appropriate decoder. MPEG-4 provides an additional Synchronization

Layer, which can add information on when each object should be composed to the final scene. An appropriate scheduler should guaranty the process of each packet in time.

Additionally, sometimes it is not efficient to use a different physical connection for each media stream. Especially, in case where MPEG-4 streams are involved, there might be some streams, which are not making efficient use of the payload space in each packet. MPEG-4 provides a lightweight multiplexing tool, the FlexMux. FlexMUx can be used to multiplex many streams in one, and thus make a better use of the network. Alternatively, RTP allows also the multiplex of many streams in one packet, as long as the right payload type is defined.

3.5 Computer-generated and Natural Content Integration

The integration of computer-generated graphics with natural audio-visual objects gives an extremely powerful way to visualize information. There are many applications that can take advantage from this integration, such as tele-learning, games and video-conferences. The parametric representation of the computer-generated graphics gives a low bandwidth solution to exchange visual information. Also, the increased interactivity that is possible with those objects makes it even more attractive. On the other hand, the natural representation of objects like human faces into 3D environments provides an even more realistic creation of virtual environments.

MPEG-4 defines the way for the integration of computer-generated streams with their natural counterparts. It makes feasible the streaming of media like MPEG-4 video into a 3D scene, where the user can navigate and interact with every object. Media and actions can be shared among the participants in the virtual space. This framework provides those characteristics in any application.

3.6 Media Compression

The compression of time-continuous media streams like audio and video is required to enable their transmission over networks. Many encoding techniques can be applied to optimize the performance according to the specific requirements of each application. For example, an application might require a very low bandwidth stream, or low complexity for encoding/decoding. Of course, this might lead in low perceived quality in the

presentation of the streams. JMF provides a big set of encoders/decoders for many popular encoding formats, such as H.263, Quick Time and MPEG-1.

In addition, it is important a compression mechanism for computer-generated graphics. MPEG-4 defines an encoding technique through BIFS, where each object is encoded using a predefined binary format.

This system maintains a BIFS encoder/decoder to convert the computer-generated graphics into a streamable, compressed format.

More advanced techniques will be available from the next release of MPEG-4, the MPEG-4 Version 2. The new version allows efficient compression of 3D meshes, which can be very large objects.

Chapter 4

4 System Architecture

The development of a multimedia collaboration system is a time-consuming and costly task. In addition, the variation of resources during the operation of those systems makes impractical the development of a library-based system, which would provide a set of classes to applications. To address these problems, an adaptive framework that would provide reusable and extendible components is required. Existing design and implementation patterns can help for a faster development.

In this chapter, the framework-based architecture is presented. The use cases of the system are analyzed and the basic objects are identified. The general class architecture, as well as the interactions among those objects, is described, using UML diagrams. The big picture of the system is presented. The analysis of subsystems follows in the next chapters.

4.1 A Framework-based Architecture

A framework is an architectural pattern that provides an extensible template for applications within a domain [8]. Users (humans) can access the functionality of this framework, through applications. The system provides some sets of interfaces to applications, to enable the transparent usage of its functionality. For example, the Session Management is provided through the DMIF Interface.

In addition, the applications have to implement specific callback interfaces. Through these interfaces, the system can inform the applications asynchronously of updates and changes. For example, the system can inform an application that a new user has joined the collaboration group, and that the new user is transmitting some new media streams. The application can then request the subscription to specific streams.

This system is organized in autonomous subsystems. Every subsystem can be used by a different set of applications, either in combination with other subsystems or by itself. For example, an application might need only the Session Management subsystem, or it might also needs to use the VRML to BIFS converter tool. Thus, the architecture of the system is highly modular.

Also, the application might require specific solutions to some subparts of the system. For example, it might wants to implement a TCP-based Session Management subsystem. It can still use the Session Management components provided by this system, and extend them by adding the TCP-based communication objects.

This framework can be mainly extended in three different ways:

- Implementing an interface of the framework
- Inheriting from a class in the framework, and
- Aggregating a framework class

4.2 Use Cases

The analysis of the system starts with the translation of the system operation requirements to usage scenarios, using UML use case diagrams. A core set of system use cases is shown in Figure 4.1.

The actor (external user) of this system is the Application. The Application should be able load saved virtual environments. This scenario allows the sharing of content among different Applications. Of course, the format of the content should be the same.

In addition, the Application should be able to connect to existing communication groups, or create new ones. This way, the online collaboration among different users can be achieved. Each Application should be able to add media channels and multicast the streams to all the interested participants.

Also, the Application should be able to configure the system and the network resources. For example, it might want to have a specific encoding format for a video stream, or specific QoS requirements.

Users should be able to navigate within the virtual environments and interact with every audio-visual object. But, every action they are making should be multicast to every other user that participates in this shared world, to update their state.

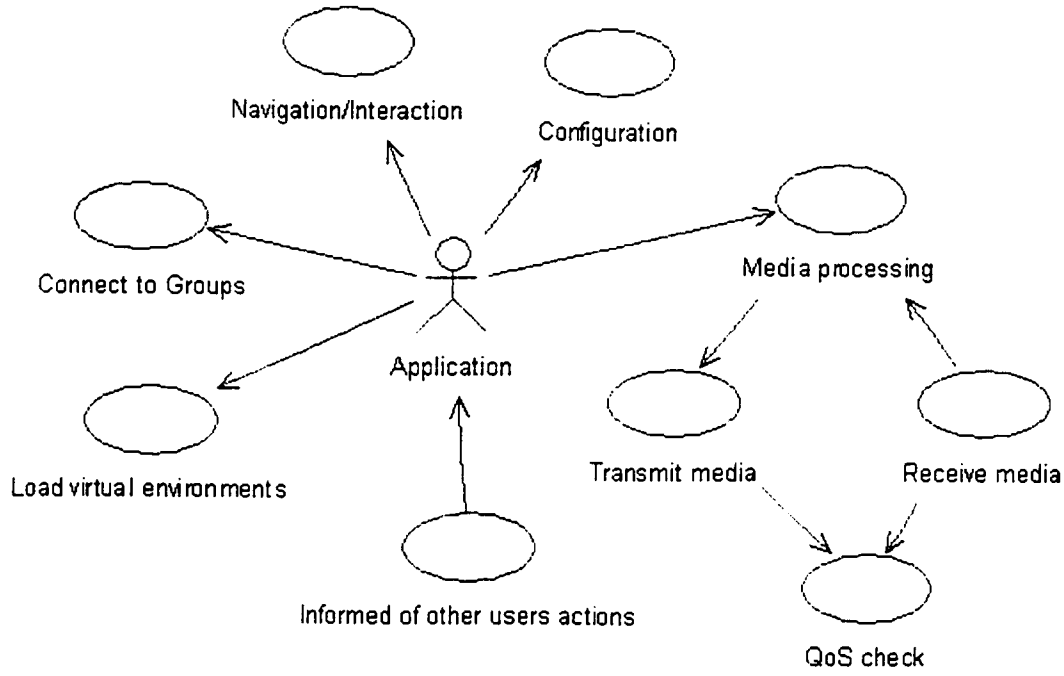

Figure 4.1 System usage scenarios

Finally, every Application should be able to process the media streams that are shared in the collaboration space and encode or decode, multiplex or de-multiplex them; or, even transcode them to other formats. Those streams are received or transmitted to other users. An optional QoS checking might also be applied to the exchanged media. If the QoS observed from a connection does not fulfill the Application requirements, then an appropriate action might be taken.

4.3 Basic Object Identification

The analysis of this system is continued with the identification of its "basic" objects. "Basic" objects are important objects, which provide the functionality of a subsystem, through a well-known interface.

The loading of a saved virtual word is responsibility of either a VRML or BIFS loader, depending on the format of the world. Those loaders implement a basic interface that provides methods to access specific files, on a local disk or from an Internet address (using a URL).

The connection to multicast groups and the management of the sessions, in general, are handled by an object, which is called DMIFFilter. The DMIFFilter provides the DMIF Application Interface (DAI) to the applications to access streams and services in multicast groups.

The navigation and interaction of each user with the virtual environment is handled by a set of behavior objects. Each of those objects is responsible to detect and handle a specific action of the user. They all derive from a base class, the Behavior class, which is part of the Java 3D API.

The configuration of the system is handled by a set of configuration-specific objects. For example, the configuration of the characteristics of the BIFS streams is handled by an object, which is called BIFSConfig. It specifies the length of specific fields into the BIFS stream to optimize the usage of the BIFS streams.

Each user is subscribed in the BIFS stream. The BIFS stream contains all the actions that every user is performing within the collaboration space. This way, every user is informed of any action of any user. Each action is encoded into a BIFS command by an object, which is called BIFSEncoder. At the receiver side, the BIFS command is decoded by another object, which is called BIFSDecoder. The SceneManager manages the SceneGraph, which implements a structure to "hold" the scene information The SceneManager is informed for each BIFS command and the SceneGraph is updated. In addition appropriate encoders and decoders exist for every stream that is exchanged among the participants in a virtual world.

Each stream is transmitted in a different session. A SessionManager is responsible for each one. It informs the application for a possible change of the encoding format of the media and when a new source, which is transmitting data, is added in the session and whether it should be handled by a different decoder. RTP-specific objects are accomplishing the transmission of the streams.

Finally, when there is a QoS violation, the application is informed through the DAI Interface. The RTPMonitor object detects the QoS violation.

4.4 General Architecture

The general architecture of the system is composed of the most important object and their relationships. Figure 4.2 shows the UML class diagram of the general architecture and 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 next chapters.

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. DMIFFilter provides the DMIF Application Interface (DAI) to applications for the control of the establishment and release of the multicast sessions. On the other hand, MPEG-4 Browser implements a DAI callback interface for the DMIFFilter. The DMIFFilter is responsible for handling all the requests for media access.

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 a directed acyclic graph. This graph is provided to the SceneManagerImpl. Similarly, the SceneManagerImpl has a BIFSEncoder and a BIFSDecoder object to create 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. They implement the Codec interface to provide their functionality. CoDec is an abstract interface for encoders and decoders (enCOder/DECoder).

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 and are retrieved from the encoders.

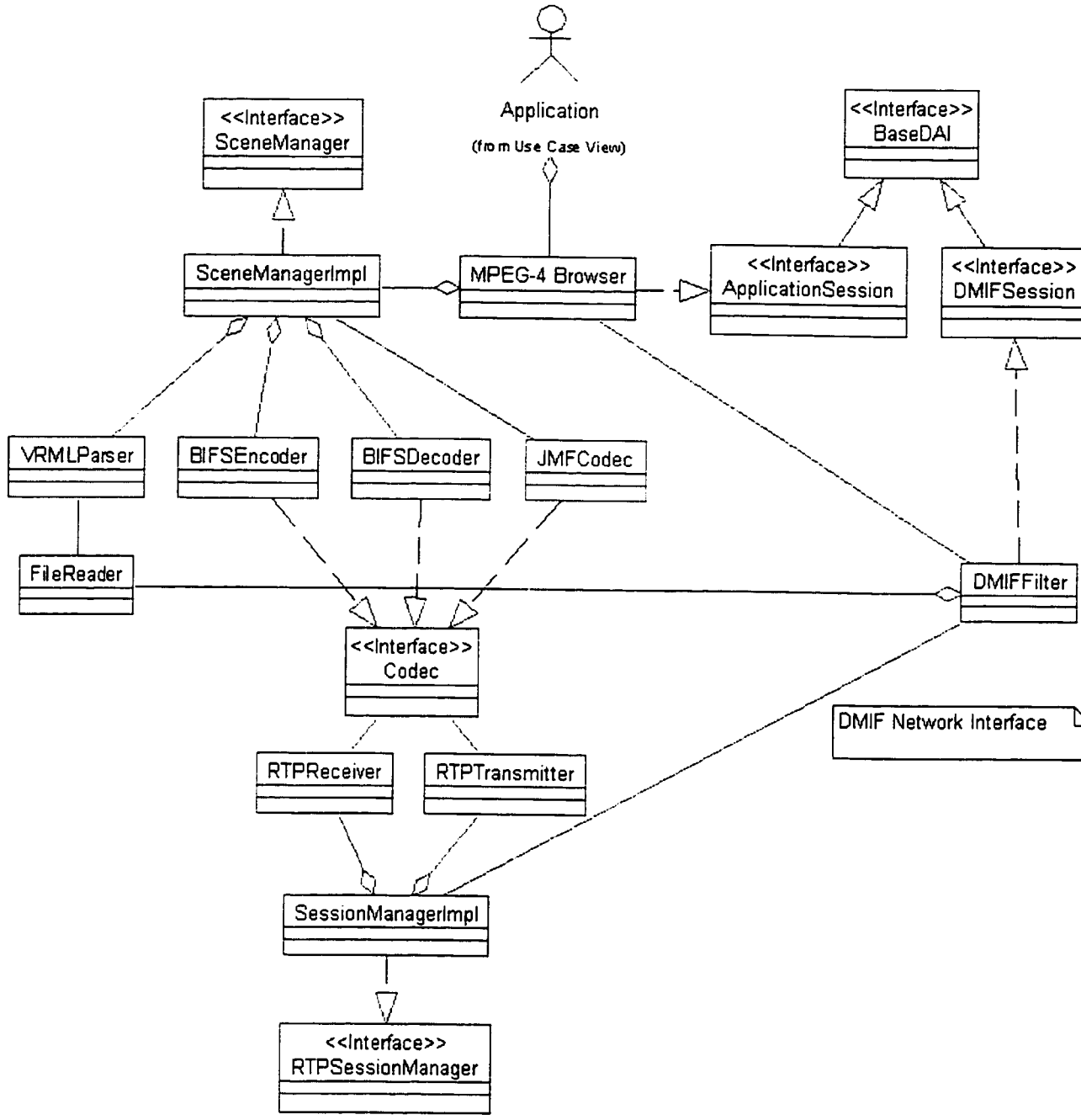

Figure 4.2 General Architecture

4.5 Simple Usage Scenarios

The UML class diagrams show the static relationship among the objects of the architecture. In this section two more different UML diagrams are provided:

- Sequence diagrams

- Collaboration diagrams

Those two types of diagrams are showing the way the objects interact with each other, at the run-time of the system. They are applied in two basic scenarios:

- Creating a producer, and
- Creating a consumer

It should be pointed out that the scenarios are not complete and in some cases there are some assumptions about the state of some objects. This way the reader can concentrate on the important details that the diagrams show.

4.5.1 Creating a Data Producer

In this scenario it is assumed that the system already knows the Multicast address and port for group session management.

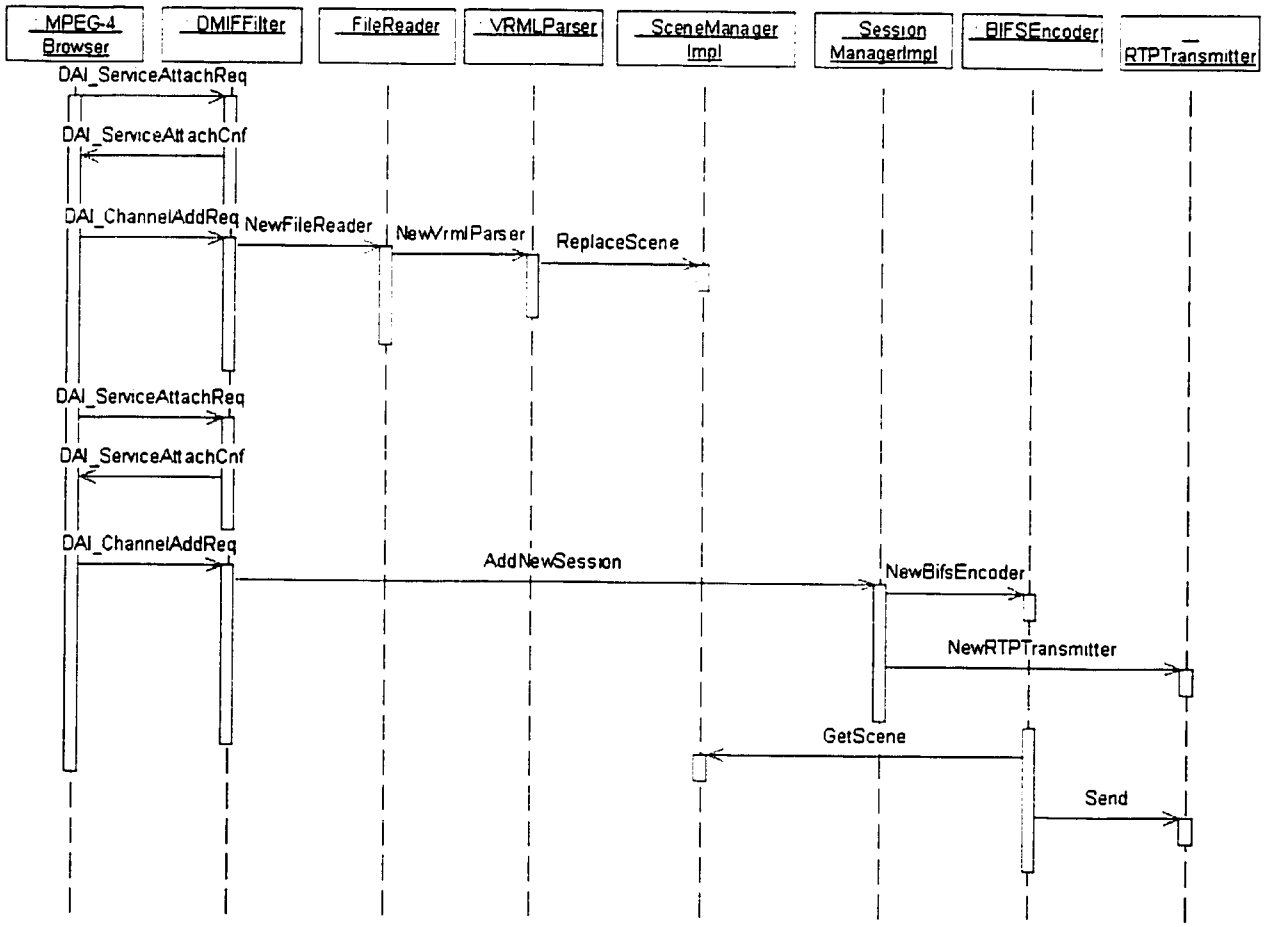

Figure 4.3 UML sequence diagram for data producer creation

Figure 4.3 shows the UML sequence diagram for creating a producer, which transmits a stream describing a scene loaded from a VRML file and encoded in MPEG-4 BIFS format. The MPEG-4 Browser is requesting from the DMIFFilter to attach to a new service that will enable the access to a local file. The DMIFFilter is informing the MPEG-4 Browser for the content of the local file, which is a VRML file. The MPEG-4 Browser is requesting the reading of that file, which is associated with the VRMLParser. The VRMLParser parses that file and creates a new scene graph. Then, it calls the ReplaceScene method of the SceneManagerImpl to replace to existing world with the new one.

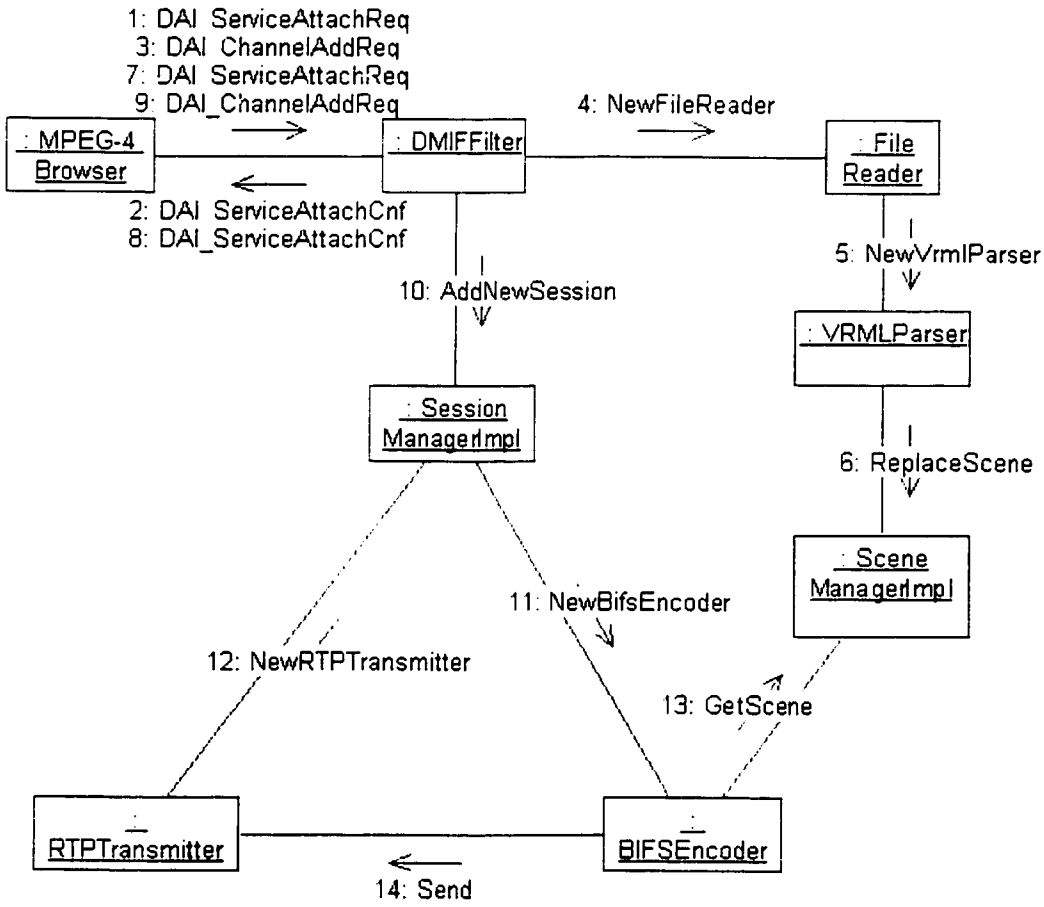

Figure 4.4 UML collaboration diagram for data producer creation

The MPEG-4 Browser is requesting from the DMIFFilter the creation of a new service for multicast transmission of the loaded VRML file. Then, it requests the addition of a new channel for the BIFS stream, which is the streamed equivalent of the loaded VRML file. The DMIFFilter adds a new session, which is handled by the

SessionManagerImpl. The SessionManagerImpl creates a new BIFSEncoder that encodes the loaded scene graph. It also creates a RTPTransmitter to handle the transmission of the stream.

A second view of the interaction among the objects of the system is giving with the Figure 4.4. It shows the collaboration diagram for the same scenario.
The steps of the scenario are numbered to show their time dependency.

4.5.2 Creating a Data Consumer

In this scenario it is assumed that the system already knows the Multicast address and port for group session management. Figure 4.5 shows the UML sequence diagram for creating a BIFS stream consumer, which receives an MPEG-4 BIFS stream and reconstructs the transmitted scene graph.

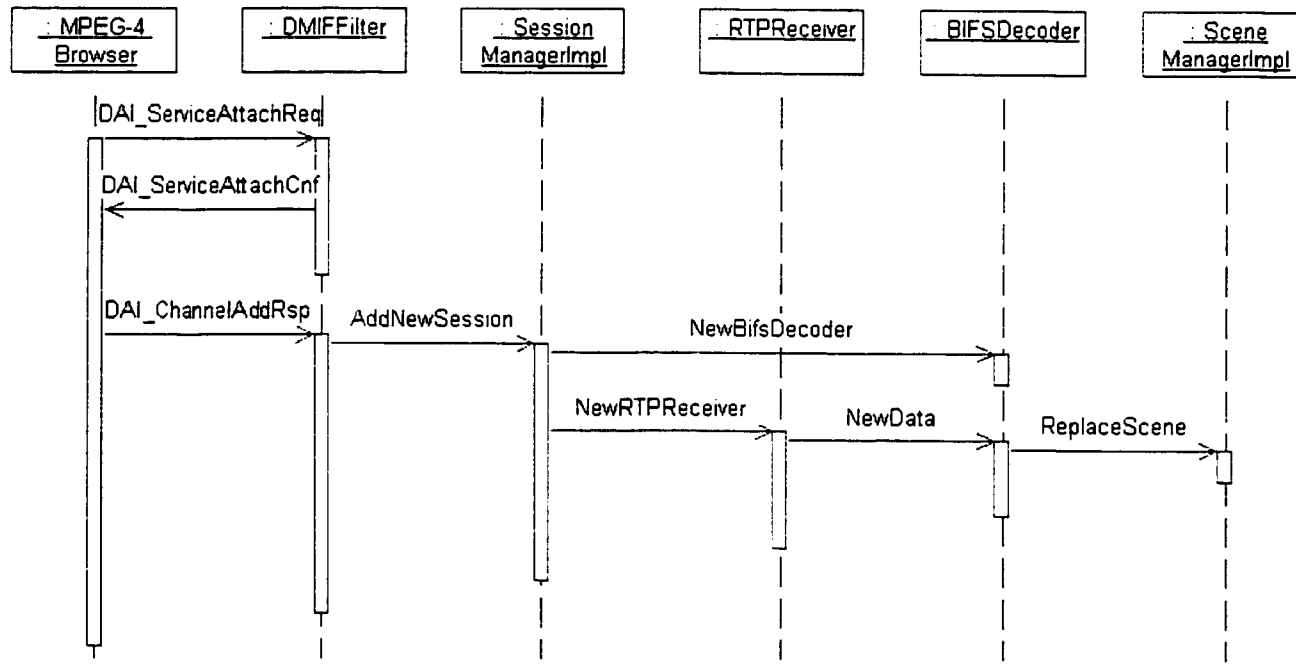

Figure 4.5 UML sequence diagram for data consumer creation

The MPEG-4 Browser is requesting from the DMIFFilter to attach a new multicast group. The MPEG-4 Browser is informed of what producers exist in this multicast group and what streams are available. Then, it requests the addition of the new channel to receive the available BIFS stream. The DMIFFilter adds a new session, which is handled by the SessionManagerImpl. The SessionManagerImpl creates a new BIFSDecoder to decode the transmitted BIFS stream. It also creates a RTPReceiver to

handle the reception of the stream. The decoded stream is translated to a scene graph structure, which replaces the scene graph at the SceneManagerImpl.

A second view of the interaction among the objects of the system is giving with the Figure 4.6. It shows the collaboration diagram for the same scenario.
The steps of the scenario are numbered to show their time dependency.

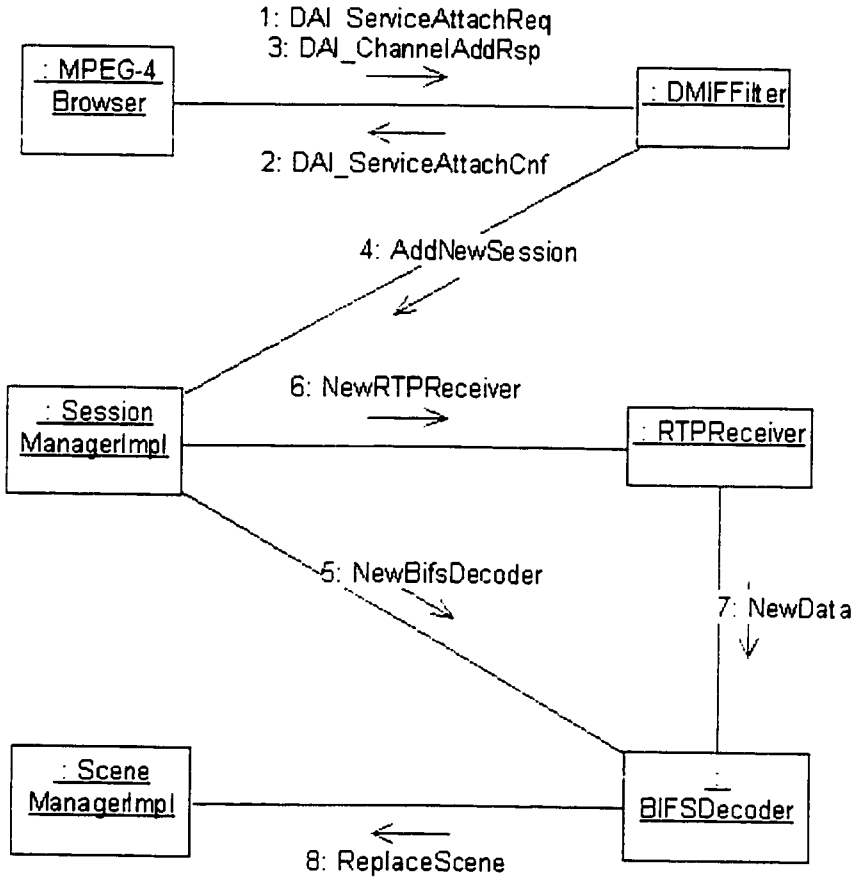

Figure 4.6 UML collaboration diagram for data consumer creation

Chapter 5

5 Scene Management and Representation

This chapter describes the way the system manages and represents the virtual scenes, which is a realization of the MPEG-4 Systems part. MPEG-4 Systems objectives are broader than the corresponding parts of MPEG-1 and MPEG-2. It extends the functionality to include advanced features such as scene description and management and 2D/3D integration, in addition to "traditional" ones, like synchronization, multiplexing and buffer management. This chapter is related to the novel features of the MPEG-4 Systems. The "traditional" concepts are described in Chapter 7.

An implementation of the MPEG-4 Binary Format for Scene (BIFS) has been maintained to compress to content of 3D scenes and enable its streaming over networks.

5.1 MPEG-4 Systems

As it has already been mentioned at Chapter 2, MPEG-4 is divided in 6 parts, the first of which is the Systems layer. While the concept of "Systems" in MPEG-1 and MPEG-2 encompasses only a description of the overall decoder architecture, the multiplexing, and synchronization mechanism, in MPEG-4, in addition to these issues, the Systems part includes scene description, interactivity, content description, and programmability.

The foundation of MPEG-4 is the coding of audio-visual objects. Examples of audio-visual objects include a video frame sequence, an audio track or an animated 3D shape. The coding of content based on objects provides many useful advantages. It allows the interaction of the user with the content, meaning that users can access, manipulate or activate specific parts of the content. Also, it improves the reusability of the content and it allows content-based scalability. Content can be adapted to match bandwidth or complexity requirements.

In order to be able to use these audio-visual objects in a presentation, additional information needs to be transmitted to the client terminals. The individual audio-visual objects are only a part of the presentation structure that is delivered to the consumers. MPEG-4 Systems defines two streams to augment the presentation of audio-visual objects at the client terminal:

- The coding of information that describes the spatio-temporal relationships between the various audio-visual objects present in the presentation content (BIFS).
- The coding of information that describes how time-dependent objects in the scene description are linked to the streamed resources actually transporting the time-dependent information (Object Descriptors).

Besides the coding of audio-visual objects organized spatio-temporally, according to a scene description, one of the key concepts of MPEG-4 is the idea of interactivity. This general idea is expressed in three specific requirements:

1. Client side interaction: The user should be able to manipulate the scene description as well as the properties of the audio-visual objects.
2. Audio-visual objects behavior: It should be possible to attach behavior to audio-visual objects. User actions or other events, like time, trigger these behaviors.
3. Client-Server or Producer-Consumer interaction: In case a return channel from the client to the server is available, the user should be able to send back information to the server that will act upon it and eventually send updates or modification of the content.

Summarizing, MPEG-4 Systems aim is to "Develop a coded, streamable representation for audio-visual objects and their associated time-variant data along with a description of how they are combined" [1].

5.2 Binary Format for Scene

5.2.1 Design Goals

MPEG-4 provides high-compression encoding for many different types of multimedia "objects": audio and speech streams, video (both frame-based as well as arbitrarily shaped), 2D and 3D meshes, still textures, synthetic audio, etc. These objects

are encoded and transmitted to the receiver in separate streams. In order to instruct the receiver on how to place these objects in space and time, MPEG-4 required a scene description capability. Binary Format for Scene (BIFS) has thus been created to support the mixing/composition of MPEG-4 objects for presentation to the user. It also enables the interaction as well as the creation, animation and modification of them in a streaming environment. BIFS is thus a central piece of the MPEG-4 object-based architecture.

As it has already been mentioned in Chapter 2, MPEG-4 considered VRML 2.0 as a basis for the BIFS format since it provided a good foundation for representation of mixed media scenes with absolute positioning. It also provided the basis for interaction and behaviors. However, it was not designed initially for inclusion in a streaming or broadcast environment. Compression and streaming are two very important goals of BIFS.

BIFS is composed of two streams:

- BIFS-Command: This is the stream responsible for progressive loading and modification of the scene.
- BIFS-Anim: This stream type is responsible for animation of the scene content. [53]

5.2.2 BIFS-command

In MPEG-4 BIFS, the presentation of the scene has a temporal dimension. In contrast to the download-based model, which is used for web applications, in the MPEG-4 model, a BIFS presentation, which describes the scene itself, is delivered over time. The basic model is that an initial scene is loaded and can receive further updates.

The mechanism with which BIFS information is provided to the receiver over time comprises the *BIFS-Command* protocol. BIFS-Command conveys commands for the replacement of a scene, addition or deletion of nodes, modification of fields, or other similar actions.

BIFS-Command protocol provides four main actions:

- Replacement of scene
- Insertion of node/field/route
- Deletion of node/value/route and
- Replacement of node/field/value/route.

5.2.3 *BIFS-anim*

When the source of an animation into a scene is a "live" one, MPEG-4 provides an alternative way to stream animation to the scene with the *BIFS-Anim* tool. BIFS-Anim enables animation, with optimal compression, for different parameters of a scene, such as positions, rotations and colors.

The BIFS-Anim framework works in three steps [53]:

1. The BIFS scene is loaded with objects that have been DEF'ed (assigned a unique node ID).
2. An Animation mask is loaded after, containing the list of nodes and fields to be animated. When multiple fields are animated, it is possible to select the field to be animated.
3. The animation stream itself is streamed containing animation frames in time-stamped access units.

BIFS-anim is not included in the implementation of the system.

5.2.4 *Scene Structure and Components*

MPEG-4 uses the VRML nodes, fields and events types to represent the scene elements. A node typically represents an object in the scene. MPEG-4 scenes are composed of a collection of nodes, arranged in a hierarchical tree. MPEG-4 has about 100 nodes. Each node consists of a list of fields that define the particular attributes of the node. For example, a Box node has a size field, which describes the length, the width and the height of a rectangular parallelepiped box.

There are four different types of field:

- *field type*, whose values are set only when instantiating the node.
- *eventIn* type, that can receive incoming events
- *eventOut*, that generate events
- *exposedField*, that may set values but also receive or generate events

An additional important feature of the node and field structure is that node fields receive default values. When instantiated, the default values are assumed, if they are not specified explicitly in the bit stream.

The scene, described with BIFS, is constructed as a directed acyclic graph of nodes, similarly to the structure of VRML scenes. The only differences are the restrictions that in BIFS there is only one top node, and that the ROUTE statements are always in the end of the stream.

5.2.5 *Events, Interaction and Behaviors*

The event model of BIFS uses the VRML concept of ROUTEs to propagate events between scene elements. ROUTEs are connections that assign the value of one field to another field (similarly to VRML). ROUTEs can be assigned an id to enable their identification for deletion or modification.

ROUTEs combined with interpolators can cause animation into a scene. For example, the value of an interpolator node can be ROUTEd to the translation field in a Transform node, causing the nodes in the Transform node's children field to be moved. User inputs can be detected with sensors. Sensors generate events when the user interacts with specific objects into the scene. A specific node, the TimeSensor, reacts to changes of time and drives the time of animations. Hence, it is necessary to have within the BIFS scene a sensor, a TimeSensor and interpolation data to perform user interaction and object animation.

5.3 *Scene Compression*

5.3.1 *General*

BIFS uses a compact representation to compress the information of the scene components. For example, when a scene parameter is specified, the minimal number of bits needed to distinguish that parameter from others is used. This scheme is used for specifying the scene contents also. For example, there are 18 different components that are used to specify the geometry of a scene: Spheres, Cones, Polygons, etc. These are indexed and specified using 5 bits. While this is not as efficient as possible, it makes parsing and specification simpler.

By default, scene parameter values are not quantized. Indeed, they are stored in their native format (e.g. 32 bits for floats and integers, 1 bit for Booleans, etc.). Moreover, the

scene parameters are classified into different categories, and the values in each category can be linearly quantized using quantization parameters (maximum and minimum values, along with the number of quantization bits) that are specified locally in the scene [53].

5.3.2 *Context Dependency*

BIFS takes advantage of context dependency in order to compress the scene efficiently at the node and the field level. Whereas VRML treats all nodes as being of type SFNode, BIFS makes use of the fact that only certain nodes can appear as children of other nodes. This allows the binary specification of the children nodes to be more efficient.

BIFS introduces the concept of a Node Data Type (NDT). There are some 30 different NDTs, and each node belongs to one or more of them. The NDT table consists of a list of nodes for each NDT. In each node data type, the node receives a fixed-binary-length local ID value that corresponds to its position in the NDT table. For example, the Shape node can appear in both a 3D context and a 2D context. It thus has both the SF3DNode and SF2DNode node data type. There is a global NDT, which is called SFWorldNode. In a 3D context, the Shape node can appear in the children field of a Transform node – this field has the type MF3DNode and it accepts nodes of type SF3DNode. The binary ID code for a Shape node thus depends on the context in which it appears. When it's a SF3DNode, it has a 6-bit binary ID value of 100100 (decimal value 36), because it occupies position number 36 in the list of 48 total SF3DNodes. Appendix B provides examples of NDTs.

This node representation is almost optimal in an information theoretic sense. It doesn't make sense to apply entropy coding techniques based on the probabilistic distribution of nodes in scenes, since such data does not currently exist [53].

5.3.3 *Node Coding Tables*

Each node is associated with a Node Coding Table (NCT). Each node's NCT holds information related to how the fields of the node are coded. The NCT specifies the type of values that each field can hold. Thus, for example, the NCT for the Transform node specifies that its children field has type MF3DNode and thus can hold a list of SF3DNode

nodes. The NCT also specifies the other field types that can occur, for example, SFFloat, SFInt32, etc.

The NCT also specifies an index for each field according to one of four usage categories. These categories are:

- DEF – used for defining field values when a node is transmitted. This corresponds to the Field and ExposedField modes, since these are the only modes that have values that can be specified.
- IN – used for data that can be modified using BIFS updates or ROUTEs. This corresponds to the EventIn and ExposedField modes.
- OUT – is used for the EventOut and ExposedField modes. That is, fields that can be used as input values for ROUTEs.
- DYN – is a subset of the IN category, used for BIFS-Animation.

By creating these 4 categories, a BIFS scene always references fields with the minimal number of bits needed. For example, the IndexedFaceSet node is used to hold a collection of polygons that form a 3D object. It has 13 fields that can be defined when it is created, and thus each field requires 4 bits when it is indexed (IN). However, this node only has 4 fields that can output values to a ROUTE, so that only two bits are needed to specify which field is being used when this node is routed from (OUT). The IndexedFaceSet node has 8 field that can accept ROUTEd values or be updated using BIFS-Command, and thus these protocols need only 3 bits to specify the modified field (IN) [53].

The NCT also defines a quantization type for each field that holds numerical values. Field quantization is not included in the implementation of this system. Appendic B provides an example of NCTs.

5.4 System Streams Relation

As it has already been mentioned, MPEG-4 Systems uses two streams to describe the scene and the relation among the streams that compose the scene:

- BIFS
- Object Descriptors

Figure 5.1 presents an example of a set of streams that compose an MPEG-4 scene.

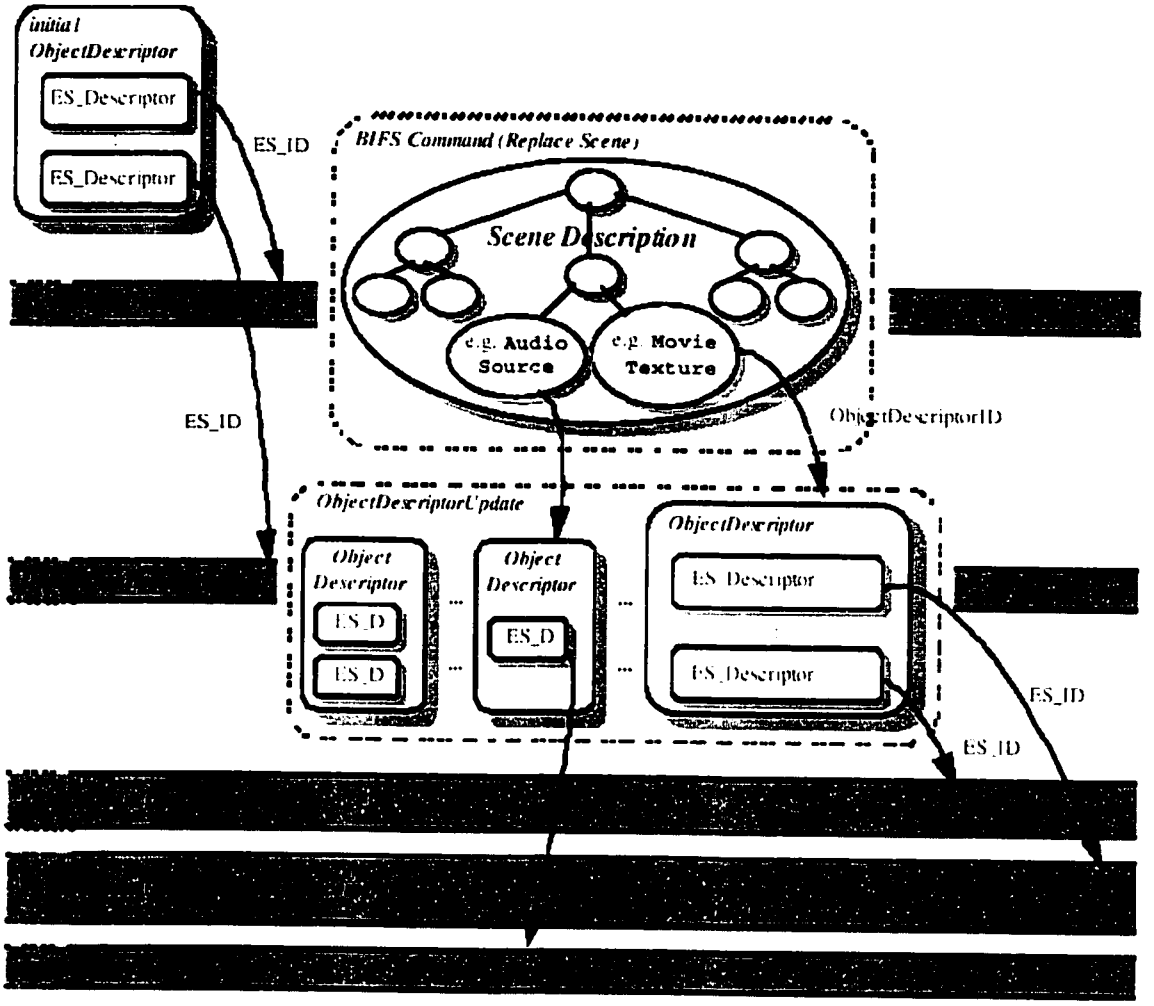

Figure 5.1 MPEG-4 Systems streams relations [29]

The top stream contains the BIFS-command stream, which updates the content of the scene. Certain nodes into the Scene Description contain pointers to Object Descriptor object of the second stream. Each Object Descriptor contains information for a specific audio or video stream. This way, an audio-visual stream can be rendered, for example, on the surface of a 3D shape.

5.5 Scene Management Subsystem Architecture

5.5.1 Subsystem Analysis

As it has already been mentioned, this system provides a framework-based architecture to applications. Applications can access the functionality of the system,

through well-known interfaces. Thus, the application is the main external actor of the system. Figure 5.2 shows the basic use-cases of the scene management subsystem.

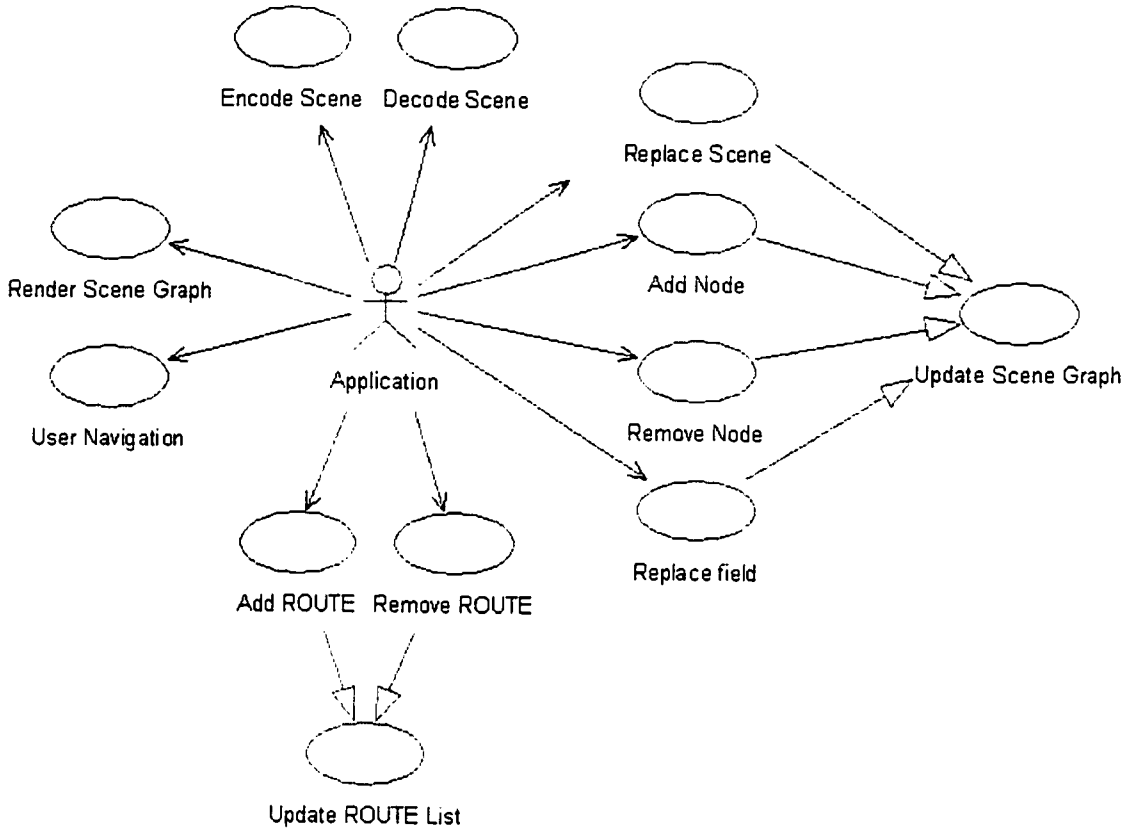

Figure 5.2 Scene management UML use case diagram

The Application can encode the contents of the scene into a BIFS stream, decode a BIFS stream and reconstruct a 3D scene. It can add or remove nodes at specific places into the directed acyclic graph, as long as the position is tagged with a specific id. Also, it can replace the contents of specific fields of nodes. More, it can add and delete ROUTEs to specify the behavior of the nodes into the scene.

In addition, the Application requires a mechanism to render the content of the directed scene graph, and to enable the navigation of the user into the 3D world with a user-friendly interface.

5.5.2 Architectural Design

The inherited complexity of the scene management subsystem is handled via a well-defined object-oriented framework, where a big number of objects are defined, with

specific responsibilities. Figure 5.3 shows the UML class diagram of the basic objects of this subsystem.

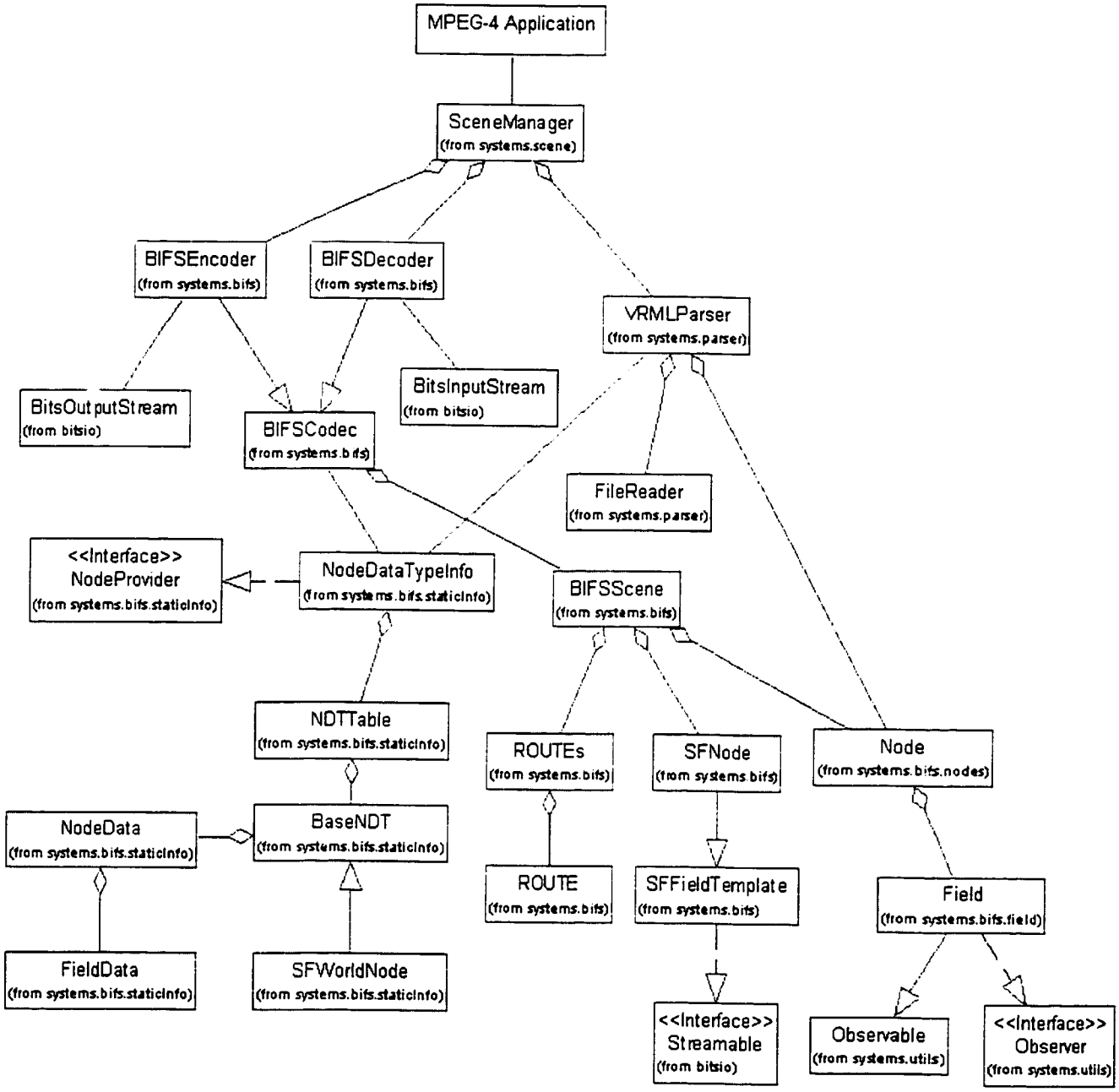

Figure 5.3 Scene management subsystem architecture

The objects that constitute the scene manager architecture are divided in seven packages.

- systems.bifs, which contains the objects which are responsible for the encoding/decoding of the objects

- systems.bifs.nodes, which provides a representation for every node
- systems.bifs.field, which provides a representation for every field
- systems.bifs.staticInfo, which contains all the objects that describe the way a specific node or a field should be encoded/decoded (NDTs and NCTs).
- systems.scene, which contains the SceneManager object
- systems.parser, which contains the objects that parse VRML text files and constructs the directed acyclic graph
- systems.utils, which provides auxiliary objects and interfaces, such as implementation of software patterns like the Observer pattern.
- bitsio, which provides a set of two objects to handle the input and output of streams, supporting bits operations.

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. They both derive from the abstract class BIFSCodec.

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 (NDTs and 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. NDTTable is a holder for NDT (Node Data Type) Tables. It identifies the NDTs and chooses the right object. All the NDT objects are derived from BaseNDT. SFWorldNode is a special table, which keeps the actual info for all the nodes. All the other tables are referencing to this table to get the requested info. NodeData is a class that encapsulates the information, which is required to encode a node (an implementation of the Node Coding Tables). 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 includes one more important object, which is called BIFSScene. BIFSScene encapsulates 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 real representation of the directed acyclic graph is modeled with the Node objects and the corresponding Fields. Fields are deriving from the Observable object 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 becoming very easy.

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.

5.5.2.1 Nodes

The representation of the nodes in the system is described using a UML class diagram, shown in Figure 5.4.

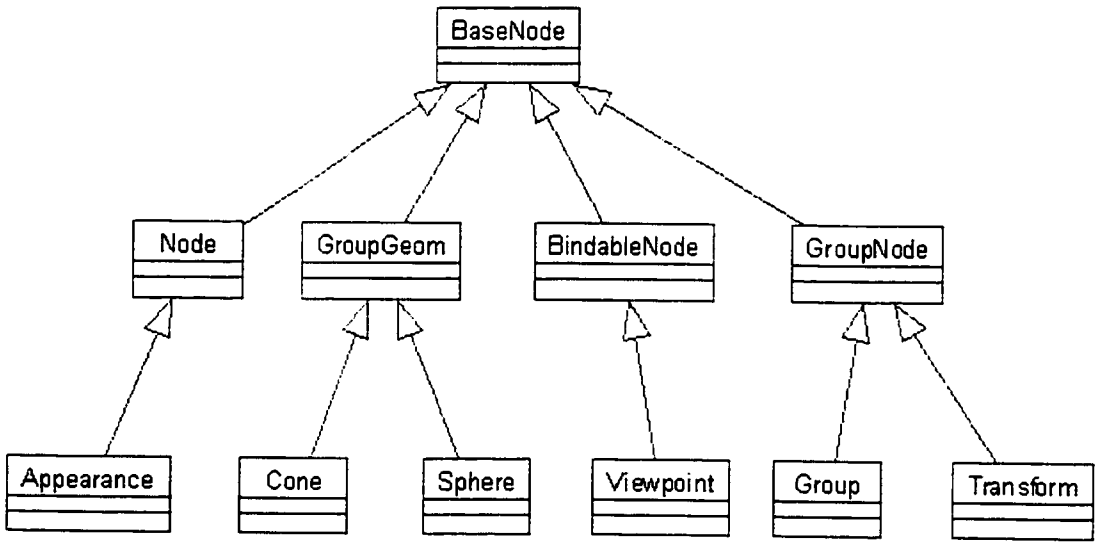

Figure 5.4 Nodes inheritance UML class diagram

BaseNode is an abstract class that every node derives. Under this node there is another layer of abstract class, which is responsible to group related nodes. For example, the GroupNode is the parent of the Group and Transform nodes, which are the implementations of the MPEG-4 corresponding nodes. The same way, BindableNode represent nodes that only one instance of them should be active at every time. For example, only one Viewpoint node is active for a specific 3D frame. Similarly, the GroupGeom node is an abstract class for all the nodes that describe geometry, such as Cone, Sphere, Box or IndexedFaceSet. Finally, there are nodes that can not be grouped in a way similar to the previous nodes. For example, the Appearance node describes the appearance of a Shape node. For this reason, one more abstract node has been defined the Node object, to include the rest of the nodes.

It should be noted that this figure does not show all the implemented nodes, but just a small subset to describe the idea of grouping and inheriting.

5.5.2.2 Fields

The same architecture model has been adapted for the description of the fields of a node. Figure 5.5 presents the idea.

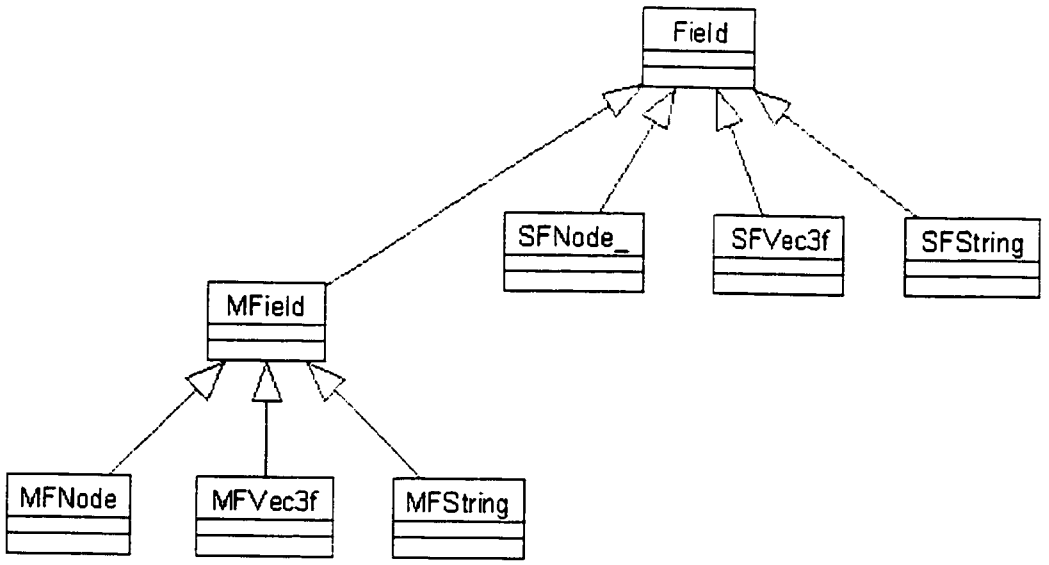

Figure 5.5 Fields inheritance UML class diagram

Field is an abstract class, which is the base for every class that models a field. A field can be either single-value or multiple-value. For the multiple-value fields, one more abstract class is defined, which is called MField. Every multiple-value field begins with the prefix MF, while every single-value field begins with the prefix SF. Example of single-value fields are the SFNode, the SFVec3f and SFString, while multiple-value fields are MFNode, MFVec3f and MFString. Every multiple-value field includes a list of single value fields. Specialized methods provide very efficient memory management, especially for very large length multiple-value fields.

It should be noted that this figure does not present every field of the model, but just a small subset to show the relation of the field classes.

5.5.2.3 Commands

Another important set of objects are those that models the commands of the BIFS-command stream. Figure 5.6 shows their relationship.

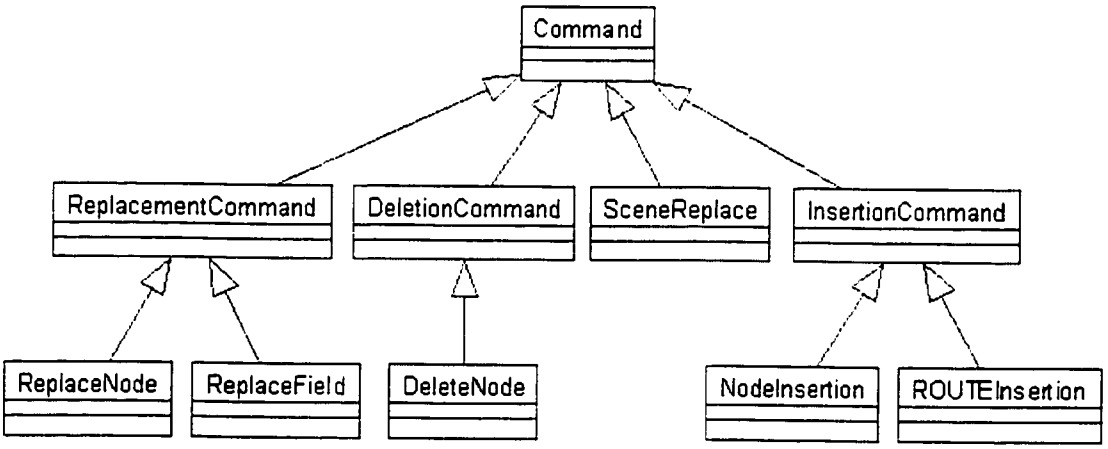

Figure 5.6 Commands inheritance UML class diagram

The Command class is an abstract class for the BIFS commands. There are four classes that inherits the base class:

- ReplacementCommand, which is an abstract class for replacement commands
- DeletionCommand, which is an abstract class for deletion commands
- SceneReplace, which replaces the scene
- InsertionCommand, which is an abstract class for insertion command

Every abstract class of the second layer is derived by classes that represents specific commands. For example, the ReplacementCommand is derived from the ReplaceNode and the replaceField command, while the InsertionCommand from the ROUTEInsertion command.

It should be noted that this figure does not present all the implemented commands of the BIFS stream. It shows a subset to describe their relationships.

5.6 Usage scenarios

The usage scenarios of this subsystem are described using UML collaboration diagrams, instead of UML sequence diagrams, which requires wider pages to be described. Every scenario is showing the simplest possible case to minimize the complexity, and some assumptions have also been made. In addition, some intermediate steps have been omitted. Each scenario is focused on the most important steps.

5.6.1 Replace Scene

This scenario presents the way the system decodes a received BIFS-command stream and reconstructs a new scene. Figure 5.7 shows the corresponding UML collaboration diagram.

The following list describes every step of the scenario:

1. The MPEG-4 Application requests from the SceneManager to replace the scene, that is contained in a stream that is passed as an argument.
2. The SceneManager uses the encapsulated BIFSDecoder to decode the stream.
3. BIFSDecoder opens the BitsInputStream.
4. BIFSDecoder identifies the BIFS command (replaceScene in this case).
5. BIFSScene is called to decode the rest of the stream.
6. BIFSScene is "rooted" by a Group node. A SFNode template object is called to decode the top node.
7. SFNode "reads' the information that identifies the type of the node and requests from the static object NodeDataTypeInfo to get the NodeData that describes how to decode this specific node.

8. NodeDataTypeInfo requests the NDT table that is related to this node (in this case it is the SFWorldNode).
9. NodeDataTypeInfo requests from the SFWorldNode the NodeData object for this node.
10. NodeData creates the required node structure (e.g. Transform).
11. NodeDataTypeInfo returns the found object to the top SFNode.
12. SFNode has the required information to decode the node. It decodes the information.
13. SFNode passes the values of the parameters of the field to the Node.
14. The Node sets the values of its fields.
15. SFNode returns the top node to the BIFSScene.
16. BIFSScene requests from the SceneManager to replace the top node, where the scen graph begins.

In this scenario, it is assumed that the new scene is composed of only one node. Recursion is used to handle the case of other nodes, which are children of the top node.

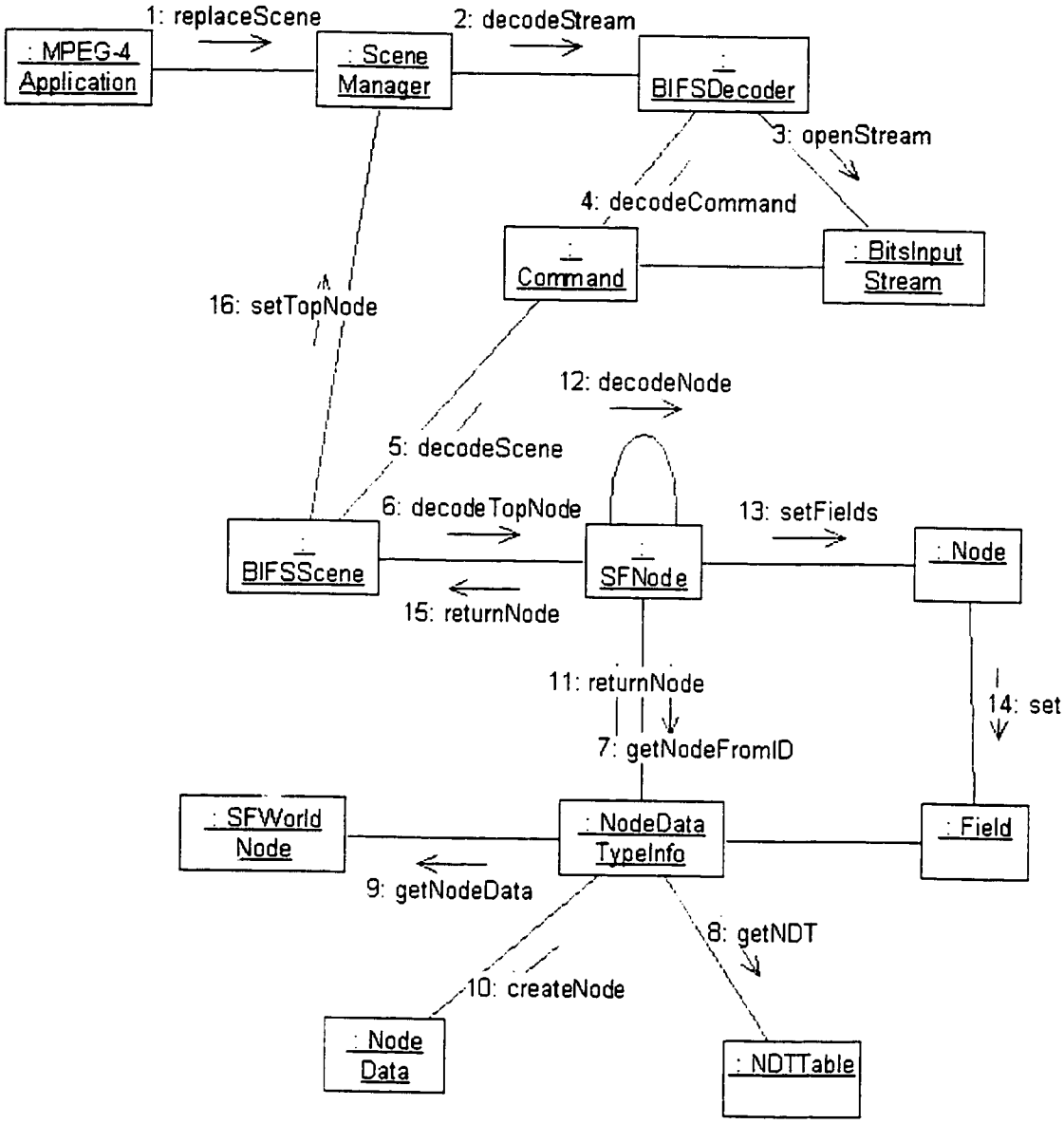

Figure 5.7 Replace Scene UML collaboration diagram

5.6.2 Add Node

This scenario presents the way the system decodes a received BIFS-command stream, which contains a command to add a node at a specific position in the graph. Figure 5.8 shows the corresponding UML collaboration diagram.

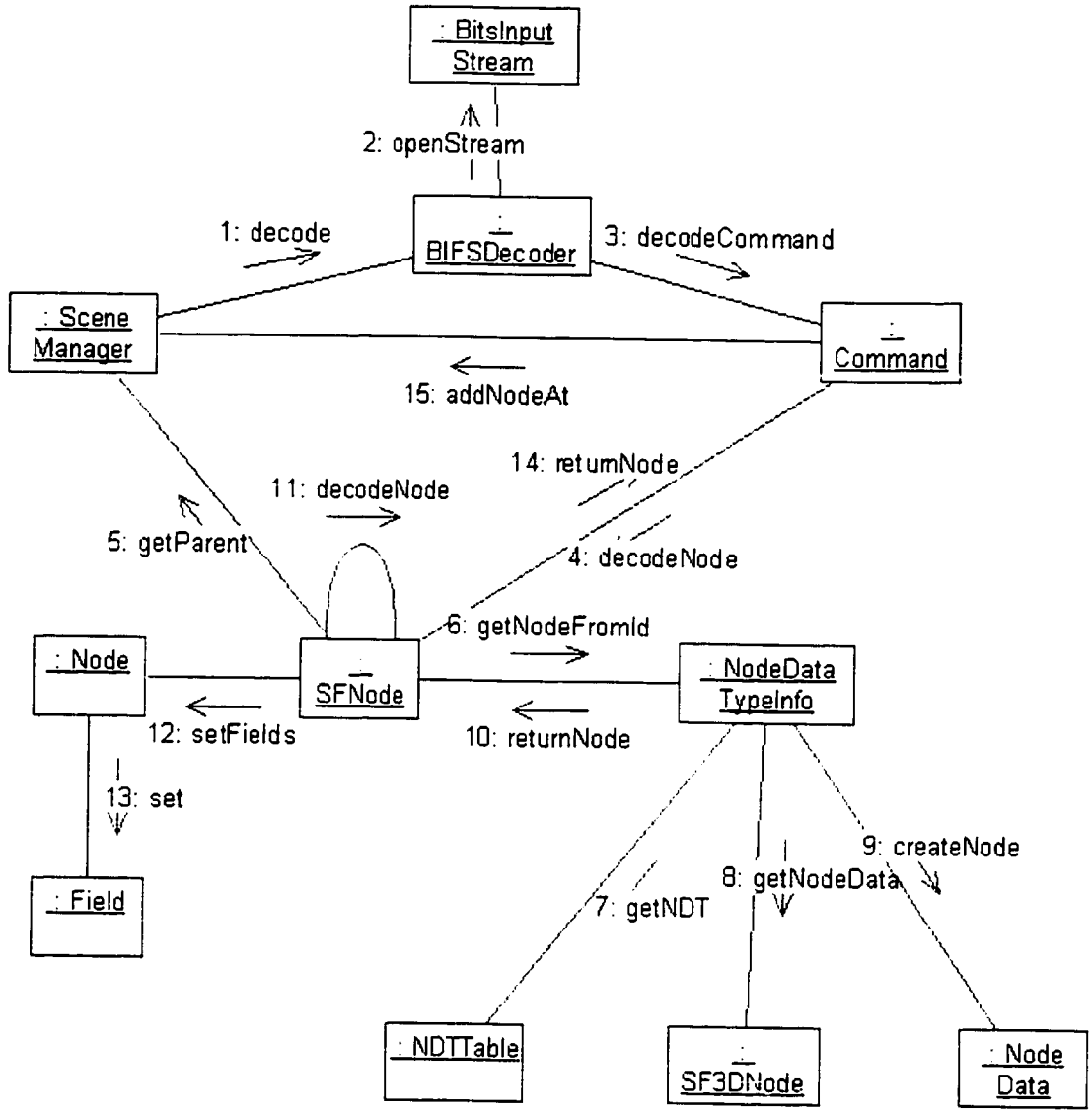

Figure 5.8 Add Node UML collaboration diagram

The following list describes every step of the scenario:

1. SceneManager requests from the BIFSDecoder to decode the received BIFS-command stream.
2. BIFSDecoder opens the received stream.
3. BIFSDecoder identifies the received command (in this case InsertNode) and requests the decoding of the stream.
4. Command requests from SFNode to decode the new node.

5. SFNode requests the parent node from SceneManager, where the new node will be inserted (to know which NDT to use).
6. SFNode requests from the NodeDataTypeInfo to identify the node from the provided id.
7. NodeDataTypeInfo requests the right NDT table to look for the node (e.g. SF3DNode).
8. NodeDataTypeInfo requests the appropriate NodeData object, which is responsible for the decoding of this node.
9. NodeDataTypeInfo requests the creation of a new Node (e.g. Shape) from the NodeData object.
10. NodeDataTypeInfo returns the created object to SFNode.
11. SFNode decodes the node from the stream.
12. SFNode requests from the Node to set its Fields.
13. Node set its fields.
14. SFNode returns the new Node to the instance of the Command.
15. Command requests from the SceneManager to at the new node at the appropriate position.

Similarly, the system acts to set the value of a field of a node, or in case of a deletion of a node.

5.6.3 Add ROUTE

The scenario of inserting a new ROUTE in the list of ROUTEs is described in this subsection. Figure 5.9 shows the UML collaboration diagram.

The following list describes the steps of this scenario:

1. SceneManager requests from BIFSDecoder to decode the new stream.
2. BIFSDecoder opens the new stream.
3. BIFSDecoder identifies the new command (in this case ROUTEInsertion) and requests the decoding of the command
4. Command requests from ROUTEs to decode the ROUTE, which is included in the rest of the stream.
5. ROUTEs creates a new ROUTE object.

6. ROUTEs requests from the new ROUTE to decode the rest of the stream.
7. ROUTE requests the node that outputs the event in the ROUTE (Out node).
8. ROUTE requests the node that receives the event in the ROUTE (In node).
9. ROUTE requests the field of the Out node that participates in the ROUTE.
10. ROUTE requests the field of the In node that participates in the ROUTE.
11. ROUTE checks the type of the two fields (it should be the same).
12. ROUTE requests from the OutField of the Out node to add the INField of the In node at the list of the Observers.
13. ROUTe requests from the InField of the In node to enable the updates from the OutField of the Out node.
14. ROUTEs requests from the SceneManager to update the list of the ROUTE.

The scenario of deleting a ROUTE is similar.

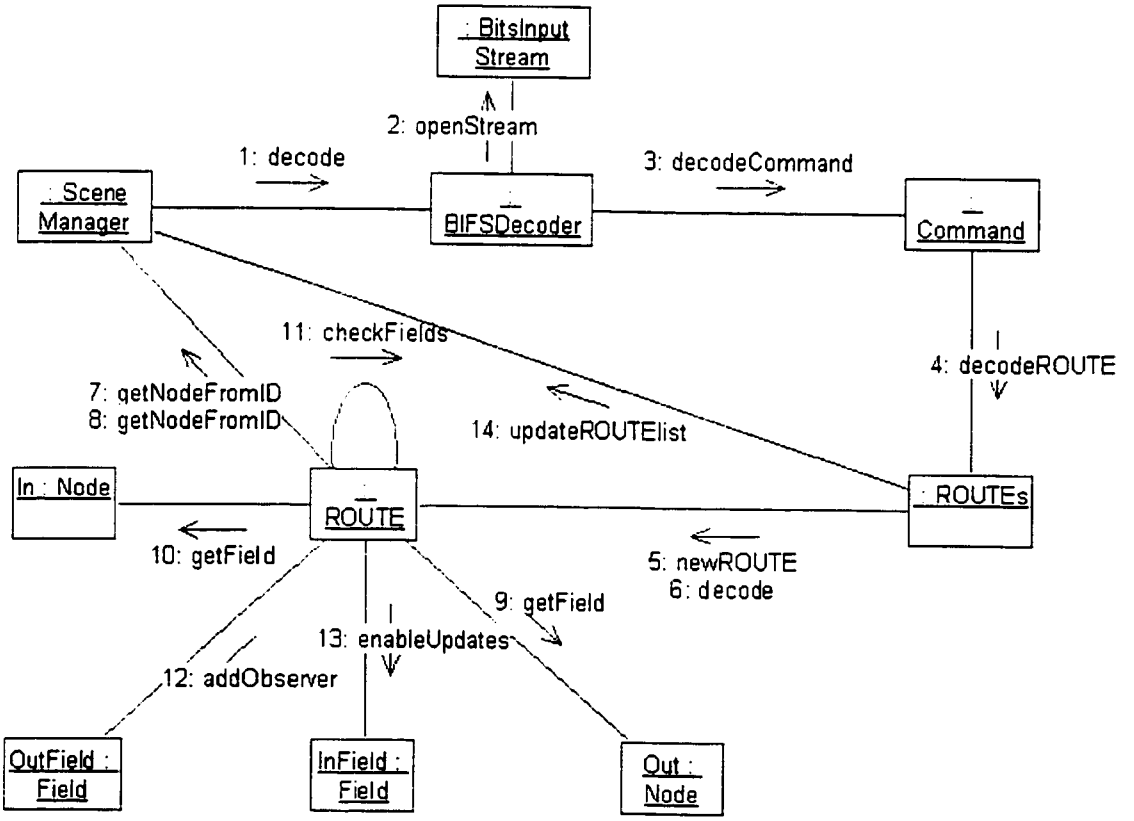

Figure 5.9 Add ROUTE UML collaboration diagram

5.6.4 *Encode Scene*

This scenario describes the way the system encodes the existing acyclic graph into a BIFS stream. Figure 5.10 shows the corresponding UML collaboration diagram.

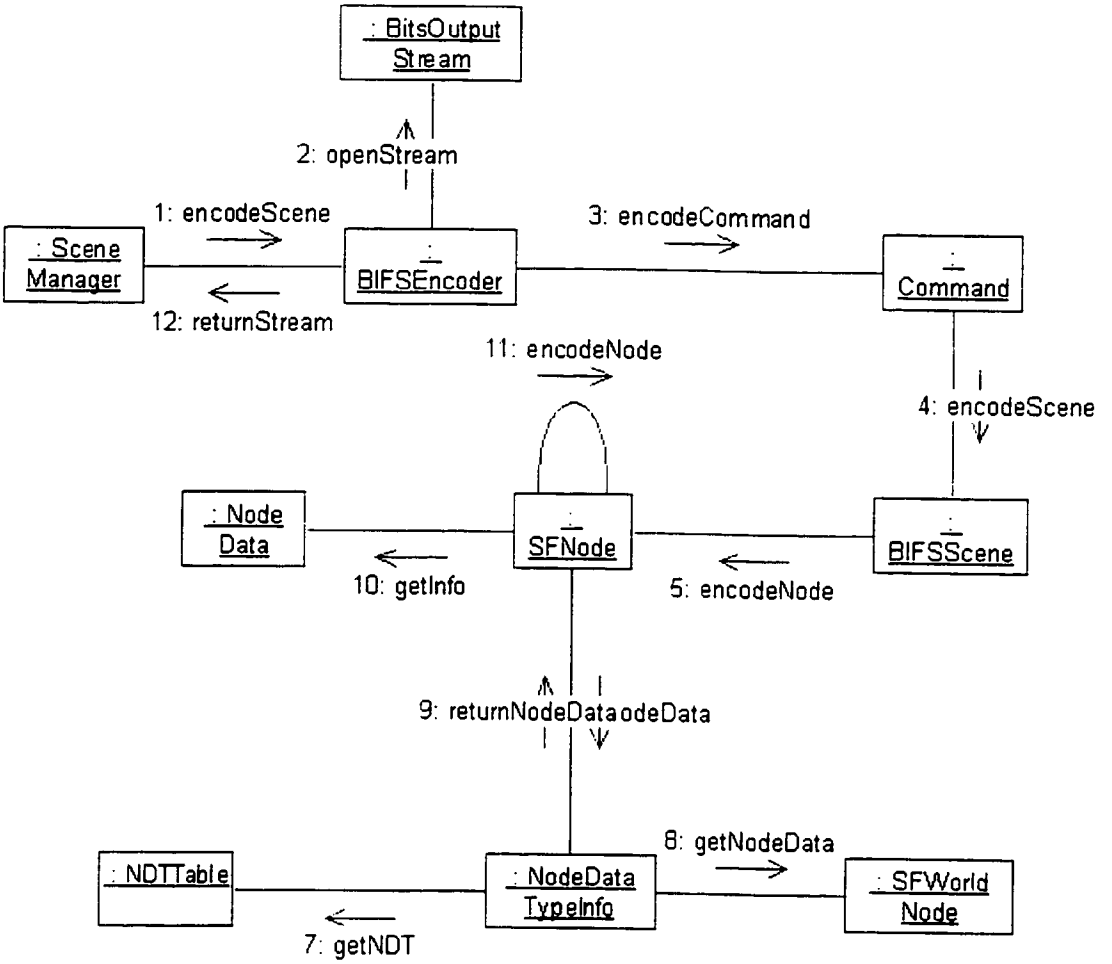

Figure 5.10 Scene encoding UML collaboration diagram

The following list describes the steps of this scenario:

1. SceneManager requests the encoding of the scene.
2. BIFSEncoder opens a new stream to write the information.
3. BIFSEncoder constructs a new Command to encode the scene.
4. Command requests from BIFSScene to encode its contents.
5. BIFSScene requests from SFNode to encode the top Node (and recursively its children).
6. SFNode requests from NodeDataTypeInfo the NodeData of the top node.

7. NodeDataTypeInfo requests from NDTTable the right NDT Table to get the info (in this case SFWorldNode).
8. NodeDataTypeInfo requests from SFWorldNode the NodeData object for the top node.
9. NodeDataTypeInfo returns the NodeData object.
10. SFNode get the information for the encoding of the top node.
11. SFNode encodes the top node.
12. BIFSEncoder returns the stream to the SceneManager.

It should be noted that the details of the encoding of the children nodes of the top node are not described. Recursive mechanisms are used for this reason.

Chapter 6

6 Session Management

This chapter describes the session management subsystem. It is based on the MPEG-4 Delivery Multimedia Integration Framework (DMIF) [30] [31], and more specifically on the multicast scenarios, which are defined at MPEG-4 Version 2.

The description begins with an overview of DMIF, where the basic information for its structure and its objectives are discussed. The communication architecture is described too, together with the two interfaces, the DAI and DNI that provides abstraction from the delivery technology and the used signaling protocol, respectively. In addition, the architecture of the subsystem is discussed as well as the multicast signaling protocol and the usage of transactions at the exchanged messages. Furthermore, the relation of the DMIF with other Internet Protocols, such as RTP is investigated. Finally, the multicast scenarios are presented to show the interaction of the objects.

6.1 DMIF Overview

6.1.1 The Problem

MPEG-4 specifies the coding of audio and video data using object description techniques. Multimedia applications compose complex scenes from one or more elementary media streams or synthetic objects. The scene description information, presented in the previous chapter, is carried in a potentially big number of data channels, especially in a multi-user environment. This implies that a big number of transport channels has to be dynamically established and released. Each transport channel requires a specific Quality of Service (QoS). In addition, the source of the transport stream can be either a broadcast source, or a multicast group, or a remote server, or even a local file.

There is a need for a system that handles the dynamic transport channel establishment and release, independently of the type of the source.

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 to modify existing applications [26].

MPEG-4 defines the Delivery Multimedia Integration Framework (DMIF) to handle the session management for the MPEG-4 terminals. It provides an interface to the applications with the required functionality to manage the establishment and the release of transport channels with QoS support.

6.1.2 DMIF into MPEG-4 Architecture

MPEG-4 terminal architecture is divided in three layers:

- The compression Layer, which performs media encoding and decoding of elementary streams. This layer is media-aware, but delivery-unaware.
- The Synchronization Layer, which manages elementary streams and their synchronization hierarchical relations. This level is both delivery and media unaware.
- The Delivery Layer, which ensures transparent access to content irrespectively of the delivery technology. This level is delivery-aware but media-unaware.

The advantage of this architecture is that every layer is focused at specific tasks and issues. For example, the compression layer is independent of the delivery mechanism and the delivery layer is unaware of the transmitted media[5].

Figure 6.1 shows the architecture of the MPEG-4 terminal. The boundary between the Synchronization and the Delivery Layer is named DMIF-Application Interface (DAI). It provides a set of methods to manage the establishment and the release of transport channels.

---

[5] This is not true for every case of transport protocol. For example, in case of transmission of media over RTP, the delivery mechanism has to know the media type and the encoding mechanism to use the appropriate payload profile.

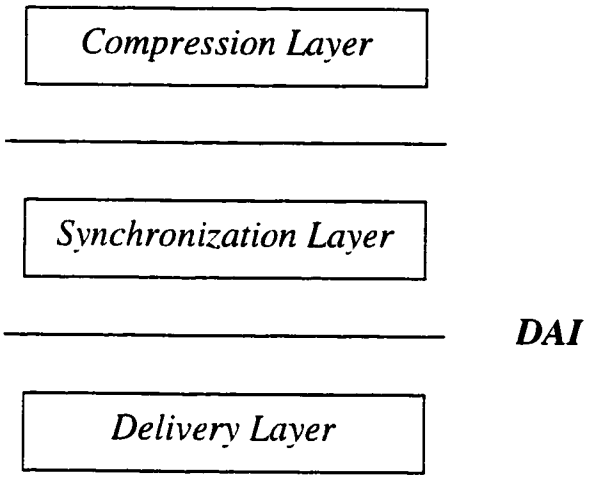

Figure 6.1 MPEG-4 terminal architecture layers

6.1.3 DMIF Objectives

The main goals of DMIF are three:

- To define a session level protocol for the management of real time, QoS sensitive streams
- To hide the delivery technology details from the DMIF User (application)
- To ensure interoperability between end-systems (in the control plane)

DMIF defines a session level protocol for the management of multimedia streaming over generic delivery technologies. DMIF is a framework more than a protocol. The functionality provided by DMIF cover the maintenance of the end-to-end QoS as expressed by the application and the efficient packing of the streams on the underlying transport technologies. The API called DAI (DMIF-Application Interface) plays an important role in isolating the application from the underlying delivery technologies by translating the commands across the DAI into protocol messages appropriate to the specific delivery technology used. These protocol messages may differ based on the delivery technology on which they operate [30].

6.1.4 *DMIF Application Interface*

The DAI is a semantic Application Programming Interface (API) that allows the development of applications irrespectively of the delivery support. No syntax is specified for this interface, since this would be programming language and operating system dependent. Furthermore, DAI preserves independence from the specific needs of MPEG-4 to enable its usage in contexts other than MPEG-4.

DAI allows the DMIF user to specify the QoS requirements for the desired stream. It is then up to the DMIF implementation to make sure that the requirements are fulfilled.

DAI is also used for accessing broadcast or multicast material and local files. Thus, by using the DAI, an application could seamlessly access content from local storage devices, from broadcast or multicast networks and from remote servers. Moreover, different delivery technologies would be hidden as well.

6.1.5 *DMIF Network Interface*

DMIF also defines an informative DMIF-Network Interface (DNI). This interface highlights the actions that DMIF shall trigger with respect to the network (e.g. to set-up or release a connection resource), 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 for remote service and multicast scenarios. DNI can also be mapped on the existing signaling infrastructures. The DNI thus describes the semantics that shall be used between DMIF peers.

6.1.6 *Computational Model*

DAI functionality is divided in three types of primitives: The Service, the Channel and the User primitives. When an application requests the activation of a service, it uses the Service primitives of DAI, and creates a service session; the DMIF implementation then contacts its corresponding peer and creates a network session with it. Network session have network-wide significance, service sessions have instead local meaning. The DMIF layer maintains the association between them. In the case of a remote interactive or multicast scenario, DMIF uses the native signaling mechanism for that network to

create and then manage the network session. The application peers then use this session to create connections, which are used to transport application data.

When an application needs a transport channel, it uses the Channel primitives of the DAI. DMIF translates these requests into connection requests, which are specific to the particular network implementation. In the case of a networked scenario, DMIF uses the native signaling mechanism for that network to create those connections. The application then uses these connections to deliver the data [30].

6.2 DMIF Communication Architecture

The DMIF architecture provides a communication architecture to applications such that they do not have to concern about the underlying communications method. The implementation of DMIF takes care of the delivery technology details presenting a simple interface to the application.

Figure 6.2 represents the above concept. An application accesses data through the DMIF-Application Interface, irrespectively whether such data comes from a broadcast or multicast source, from local storage or from a remote server. In all scenarios the Local Application only interacts through a uniform interface (DAI). Different DMIF instances translate the Local Application requests into specific messages to be delivered to the Remote Application, taking care of the peculiarities of the involved delivery technology. Similarly, data received at the terminal is uniformly delivered to the application through the DAI.

Different, specialized DMIF instances are indirectly invoked by the application to manage the various specific delivery technologies: this is however, transparent to the application, which only interacts with a single "DMIF filter". The filter is in charge of directing the particular DAI primitive to the right instance.

Moreover, the multicast and the interactive communication instances that handle the respective scenarios, invoke network-specific commands through DNI, which provides transparent access to the network specific objects.

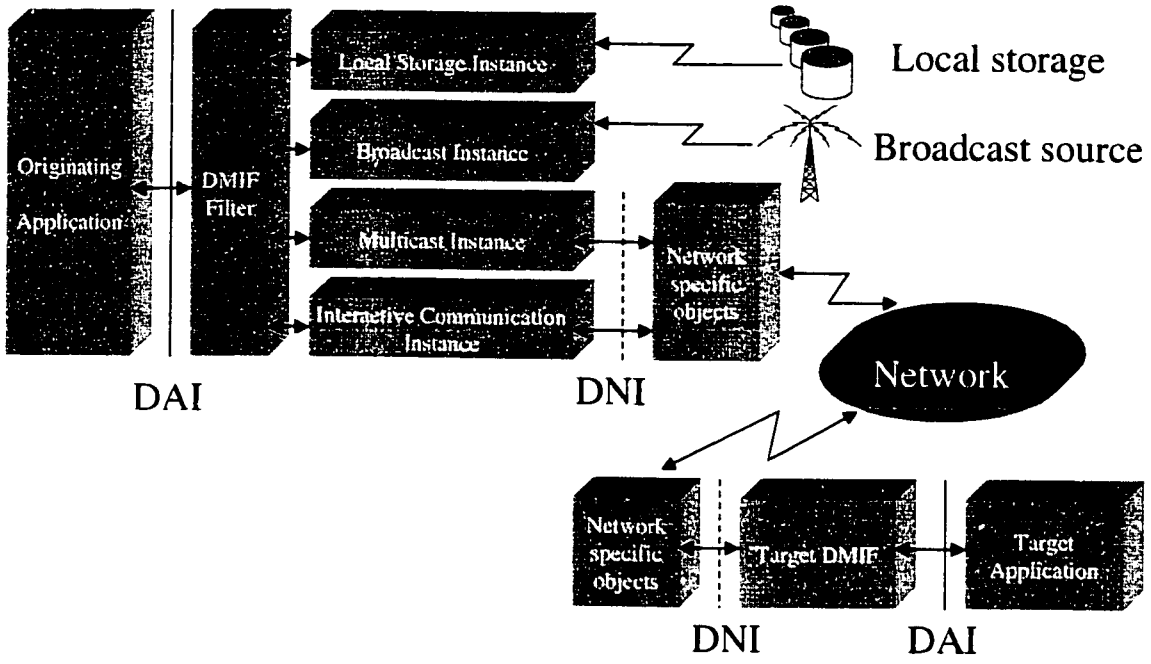

Figure 6.2 DMIF communication architecture

6.3 DMIF Multicast Architecture

The DMIF Group scenario is intended for multimedia applications that require the interaction of several DMIF terminals. As a result, it has been chosen as the solution for session management of this collaboration system. A DMIF Group session consists of a DMIF Group signaling channel to distribute the state information of the session and one or more DMIF Group transport channels to deliver the multimedia information. A DMIF terminal in a Group session can be a Data Producer DMIF Terminal (DPDT), that is, a source of information, or/and a Data Consumer DMIF Terminal (DCDT), that is a receiver of information. Applications access those objects through the DAI and the DMIF Filter. The general architecture of an MPEG-4 DMIF terminal is shown in Figure 6.3 [31].

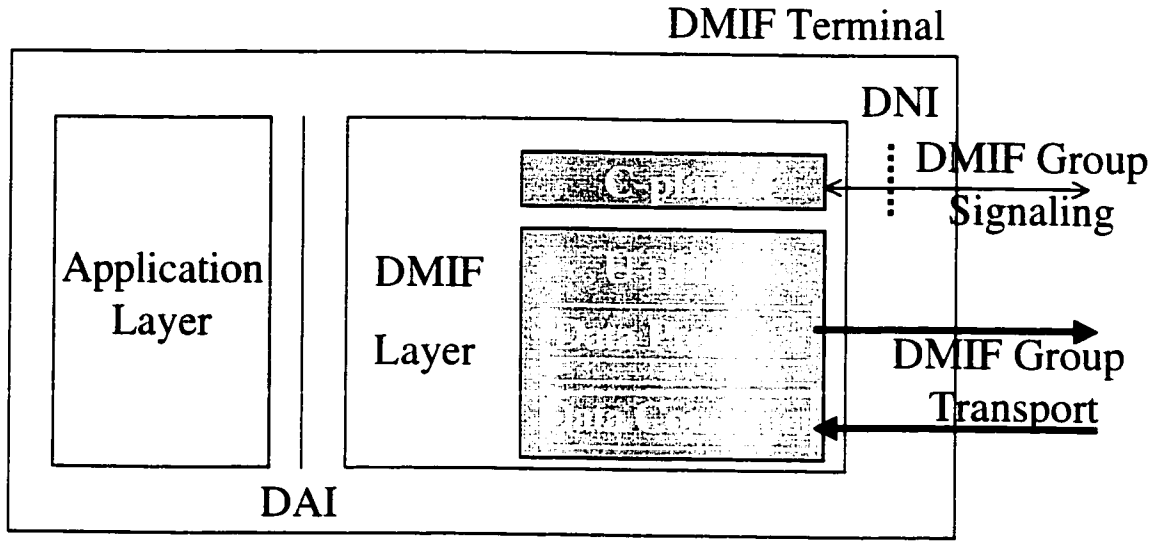

Figure 6.3 DMIF multicast architecture

6.3.1 Scenarios Analysis

The Data Producer and Data Consumer components are the basic objects in the multicast architecture. An application can be consider as Consumer if it receives media streams and producer if it transmits them. Figure 6.4 shows the UML use case diagram of the system, where actors are applications that either act as Consumers or Producers.

The basic use cases for a Producer are four:

- Join Group, where a Producer announces its existence and it reserves a service id, which uniquely identifies itself.
- Leave Group, where a Producer announces that it unsubscribes from the group.
- Create Channel, where a Producer announces the addition of a new transport channel.
- Destroy Channel, where the Producer announces the deletion of an existing transport channel.

Similarly, the basic use cases of a Consumer are the following:

- Join Group, where it requests for announcements of existing Producers.
- Leave Group, where it stops listening for the state of the Producers of a group.
- Update State, where it is informed of every change in the state of each Producer (new Producer or new channel).
- Add Channel, where it subscribes to a specific transport channel of a producer.

- Remove Channel, where it unsubscribes from a channel of a Producer.

In addition, some restrictions are applied to the group sessions:

- Data Producers must explicitly join and leave a group session.
- Only Data Producers are allowed to create and destroy channels.
- DMIF terminals are not aware of other Data Consumers that are attached to the group session.

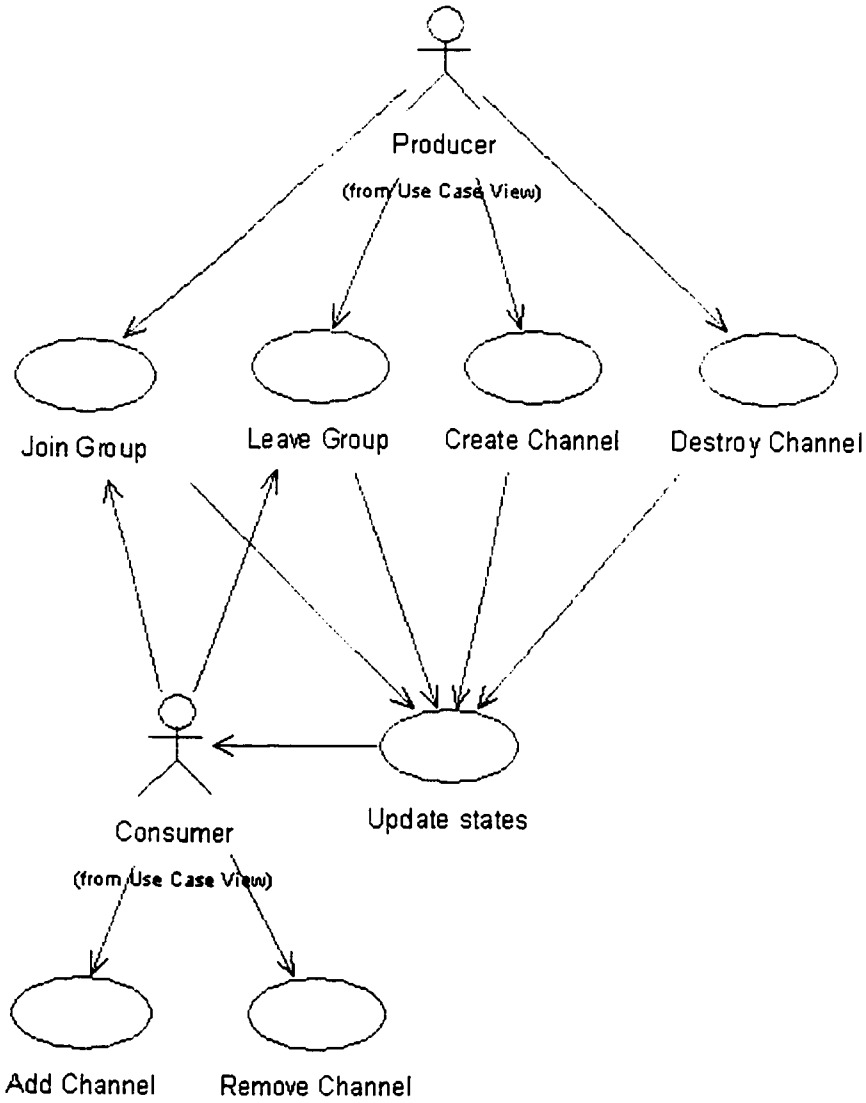

Figure 6.4 DMIF use case diagram

6.3.2 *Architectural Design*

The design of the DMIF subsystem is shown in Figure 6.5, where the basic objects and interfaces, as well as their relationships are described using a UML class diagram.

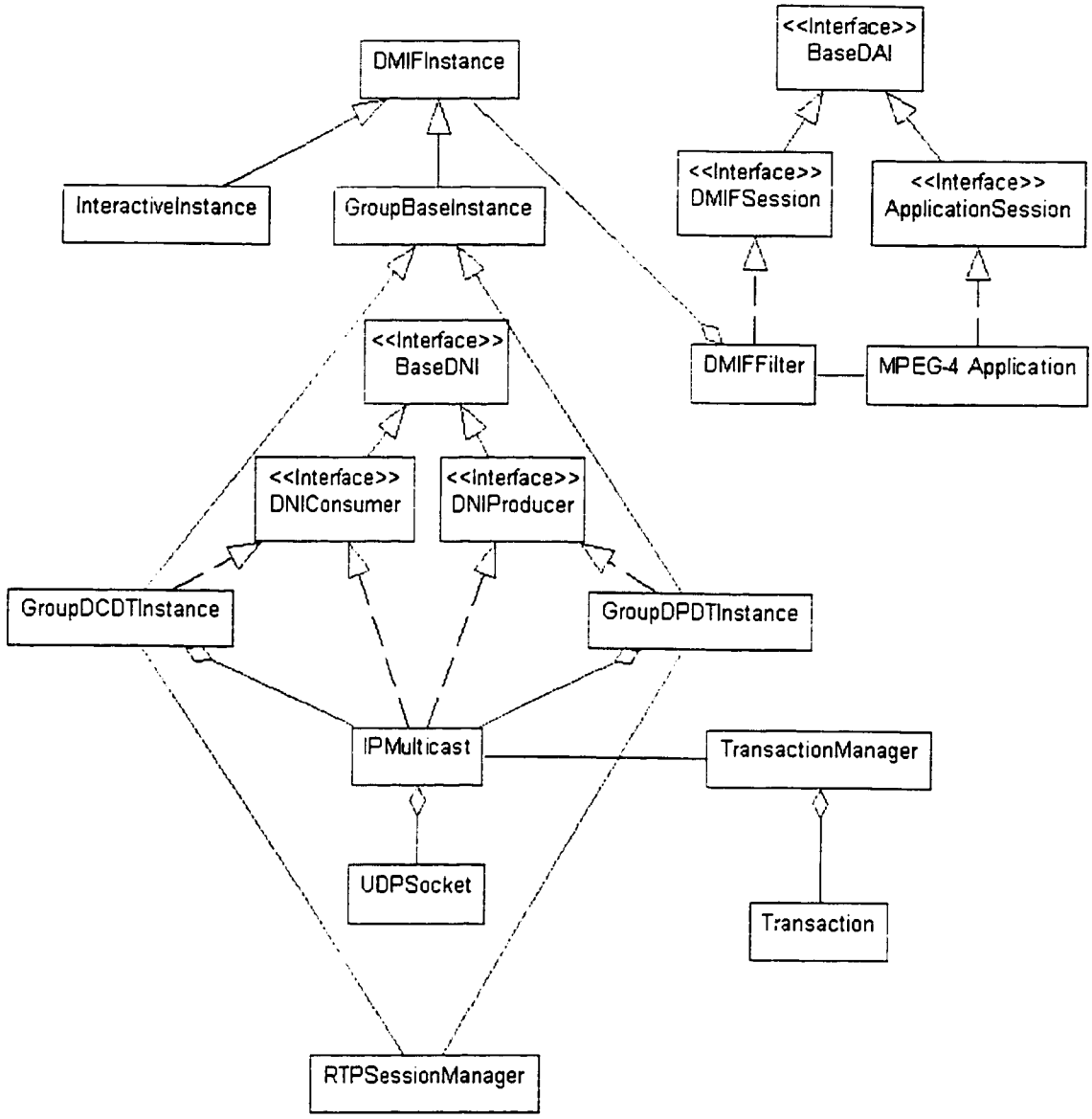

Figure 6.5 DMIF subsystem class diagram

Every MPEG-4 Application that wants to use 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 GroupDPDTInstance, 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. Each of those interfaces defines 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 create 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.
The InteractiveInstance object is an abstract class, similar to GroupBaseInstance that provides a way to extend the system to support the remote interactive scenarios.

6.4 DMIF Multicast Signaling Protocol

The signalling channel is used by Data Producer DMIF Terminals (DPDTs) to announce the creation of new transport channels and by Data Consumer DMIF Terminals (DCDTs) to query about DPDT states in the DMIF Group session. For instance, in the case of using IP technology, a group channel can be identified through an IP address and a port number.

The signalling protocol is taking part between the DPDTs and the DCDTs. IPMulticast objects are used for the exchange of the messages over the network. But, the "knowledge" of the next step of the protocol is decided in those two objects. Each one employs a finite state machine to model its behaviour.

6.4.1 *DPDT Behavior*

The behavior of a DPDT object is shown in Figure 6.6, which is a UML state diagram.

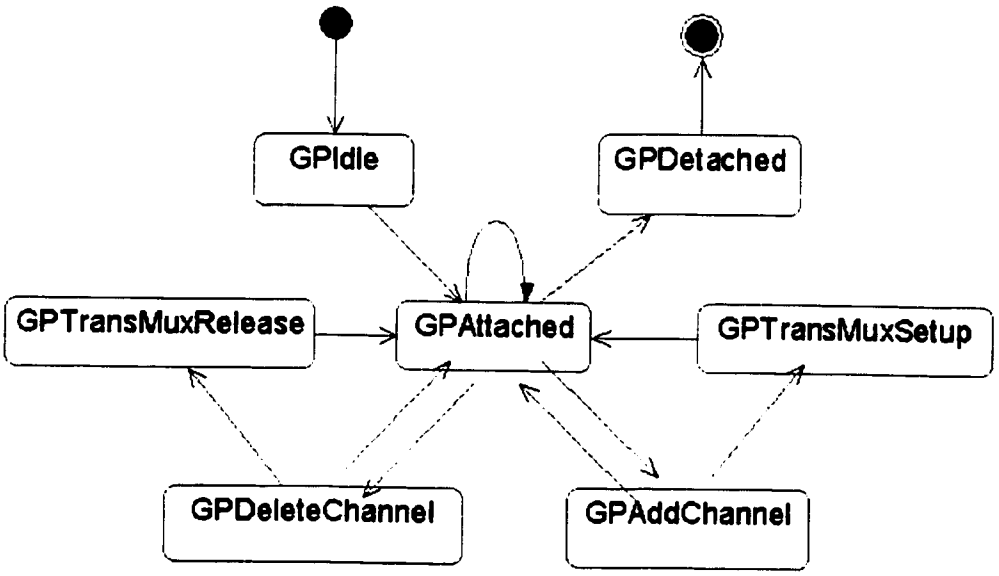

Figure 6.6 DPDT state diagram

In the beginning, a DPDT object is in the GPIdle state. After the request of an application to attach a group as a producer, it generates a unique session id and it moves to GPAttached state. Now, it is ready to accept requests for creation or deletion of new transport channels.

If the application requests a new channel, it can create a new physical connection and move to GPTransMuxSetup state and then, it can add the logical channel in this physical connection (move to GPAddChannel state). Alternatively, it can use an already existed physical (move directly at the GPAddChannel state). After the creation of the new channel, it moves back to GPAttached state.

If the application requests to destroy an existed channel, it moves to GPDeleteChannel state and then it deletes the logical channel. If no other logical channels use this physical connection, then it moves to GPTransMuxRelease to release the reserved network resources, Otherwise, it moves back to GPAttached state.

Finally, if the application requests the detaching from the group, the DPDT object announces the event and it moves to GPDetached state. It releases all the reserved resources and it ends its "life".

6.4.2 DCDT Behavior

The behavior of a DCDT object is shown in Figure 6.7, which is a UML state diagram.

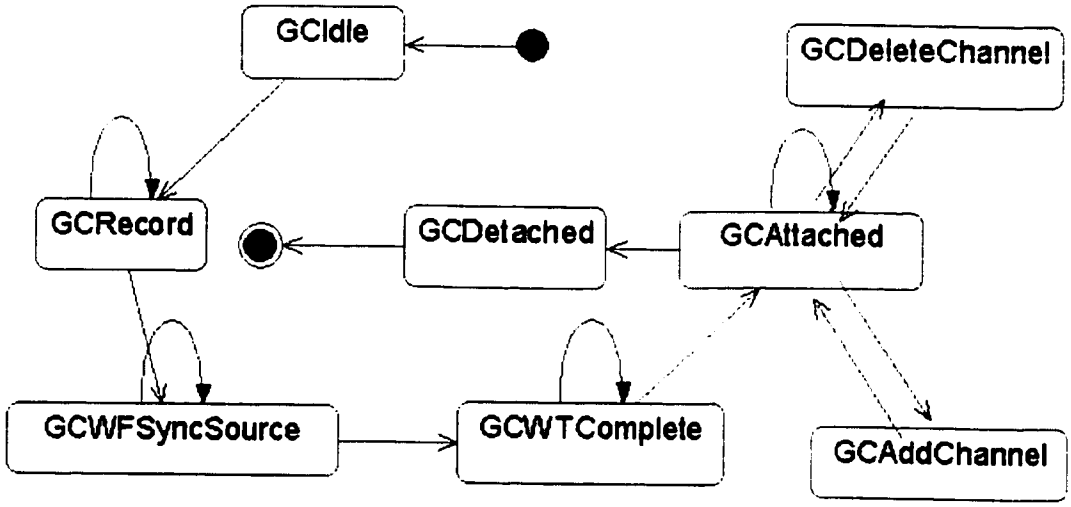

Figure 6.7 DCDT state diagram

In the beginning, the DCDT object is in the GCIdle state and it waits for the requests of the application. When the application requests the subscription to a specific group, the DCDT object moves to GCRecord state, where it listens on the signaling channel for a short period of time (depending on the round trip time - RTT). This way, it avoids requesting information that has been already requested. If the DCDT does not acquire the information within that time, it sends a SyncSourceRequest message and it moves to GCWFSyncSource state. In this state it waits for responses that contains the source id of every DPDT. After, it sends a SourceStateReuest message to the sources that it is interested in, and it moves to GCWTComplete state. In this state, it waits to receive the messages that describe the state of DPDTs that it is interested in. When, this phase is completed, it informs the application about the existing sources and channels, and it moves to the GCAttached state, where it waits for new requests.

When the application requests the subscription to a specific channel, the DCDT object moves to GCAddChannel state and it subscribes to the requested channel. After, it moves back to GCAttached state.

When the application requests to remove from a subscribed channel, the DCDT object moves to GCDeleteChannel state and it removes the requested channel from the list of the active channels. Then, it moves back to the GCAttached state.

When the application requests to leave from the group session, the DCDT object removes all the active channels and unsubscribes itself from the session group. Then it moves to GCDetached state and it becomes inactive.

6.4.3 Protocol Messages

The messages that the two DMIF peers (DCDT and DPDT) exchange to realize the multicast signaling protocol are ten. They are described in the following list:

- DS_GRP_TransMuxSetup, which announces the addition of a new physical connection for a transport channel.
- DS_GRP_TransMuxRelease, which announces the release of an existing physical connection.
- DS_GRP_SessionJoin, which announce that a new source is subscribed to the group session.
- DS_GRP_SessionLeave, which announce that an existing source is leaving from the group session.
- DS_GRP_UpstreamChannelAdded, which announce that a new logical channel is added in a physical connection.
- DS_GRP_ChannelDelete, which announce that an existing channel is removed from a physical connection.
- DS_GRP_SyncSourceRequest, which request the source id of every active source in the group session.
- DS_GRP_SyncSourceResponse, which includes the source id of a source.
- DS_GRP_SyncStateRequest, which requests the state of a specific source, identified by the source id, which is included in the message.
- DS_GRP_SyncStateResponse, which includes the state of a specific data source.

The announcement messages, as well as the response messages are sent from the data producers (DPDTs). On the other hand, the request messages are sent from the data consumers (DCDTs).

6.4.4 Transaction Management

Since the used transport protocol is unreliable (UDP), there is a need for a mechanism that ensures the reliable transmission of a request message and its response. For example, when a DCDT object sends a request for the state of a specific DPDT, it waits for the reception of the response (blocks). If either the request or the response is lost, the DCDT will block in this state forever. For this reason, there is a need for a transaction management mechanism. A pair of messages (request-response) constitutes a transaction.

As it has already been mentioned, every IPMulticast object communicates with the TransactionManager to handle the transactions of the protocol. The TransactionManager is an object that handles a list of Transaction object. It creates new ones for every message, to handle possible retransmitted messages or lost ones.

Each Transaction object is responsible for a specific transaction. In reality, two Transaction objects are required for each transaction, one at the DPDT and one at the DCDT. Figure 6.8 shows the UML state diagram of each transaction, which models the behavior of each Transaction object. Those two transactions share the same transaction id.

A new Transaction object starts at the TUndefined state. It allocates a new transaction id and if it is a request message, it moves to the TActiveRetrans state. Also, it starts a timer to detect a possible lost of the request message. If, the timer expires, then the Transaction object retransmits the lost packet, up to a maximum number of times. If no response is received after this number of retransmissions, the DPDT is considered unreachable and it is removed from the list of the active sources.

On the other hand, if the message is a response, the Transaction object moves to the TActiveNoRetrans state. If a new message arrive at the DPDT with the same transaction id, the Transaction object retransmits the response, and no more actions are taking place.

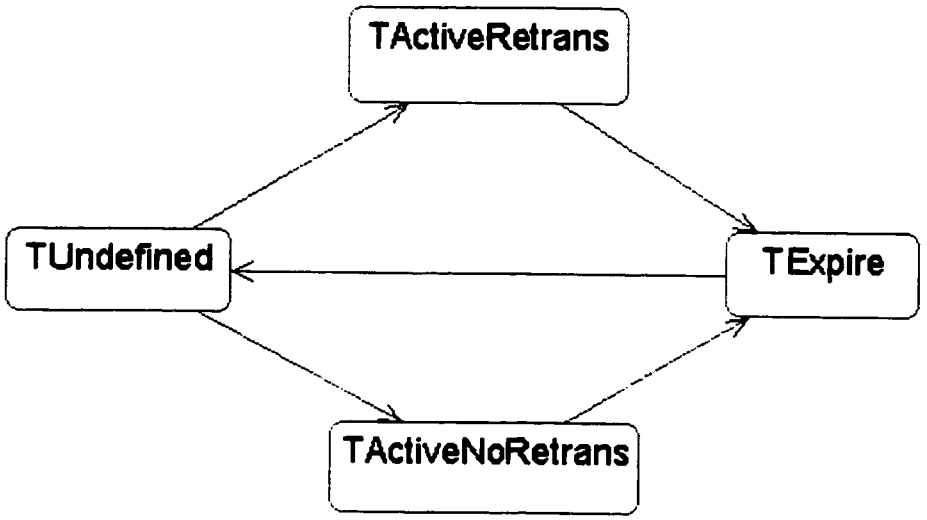

Figure 6.8 Transaction state diagram

This way, there is a guaranty for the integrity of the state of each participant of the group session[6].

6.5 Multicast Scenarios Description

In this section, the basic multicast scenarios are described using UML sequence diagrams. They do not provide all the details, but they are focused on the important interactions of each scenario.

6.5.1 Producer Service Attach

This scenario describes the messages that are exchanged among the basic objects to complete the attachment of an application as a producer (DPDT). Figure 6.9 shows the corresponding UML sequence diagram.

When an application wants to attach to a group session as a DPDT, it calls the method DAI_ServiceAttachReq from the DMIFFilter. DMIFFilter parses the URL to identify the required delivery technology and it generates the necessary information for the new request, such as a session service id. Then, it creates a new instance that is

---

[6] This transaction mechanism is inherited from the MPEG-2 Digital Storage Media – Command and Control (DSM-CC).

capable of handling the required delivery technology. In this case, it creates a GroupDPDTInstance. The GroupDPDTInstance creates a new IPMulticast object to handle the message exchange over UDP/IP for the signaling protocol. Then, the GroupDPDTInstance calls the DN_GRP_SessionJoin method to request the subscription in the multicast group as a data producer. The IPMulticast object, requests from the TransactionManager a new Transaction to handle the new message. Then, it creates the new message and it sends it across the network. The DMIFFilter informs the application about the new service session id, which the application should use in the future to refer to this session.

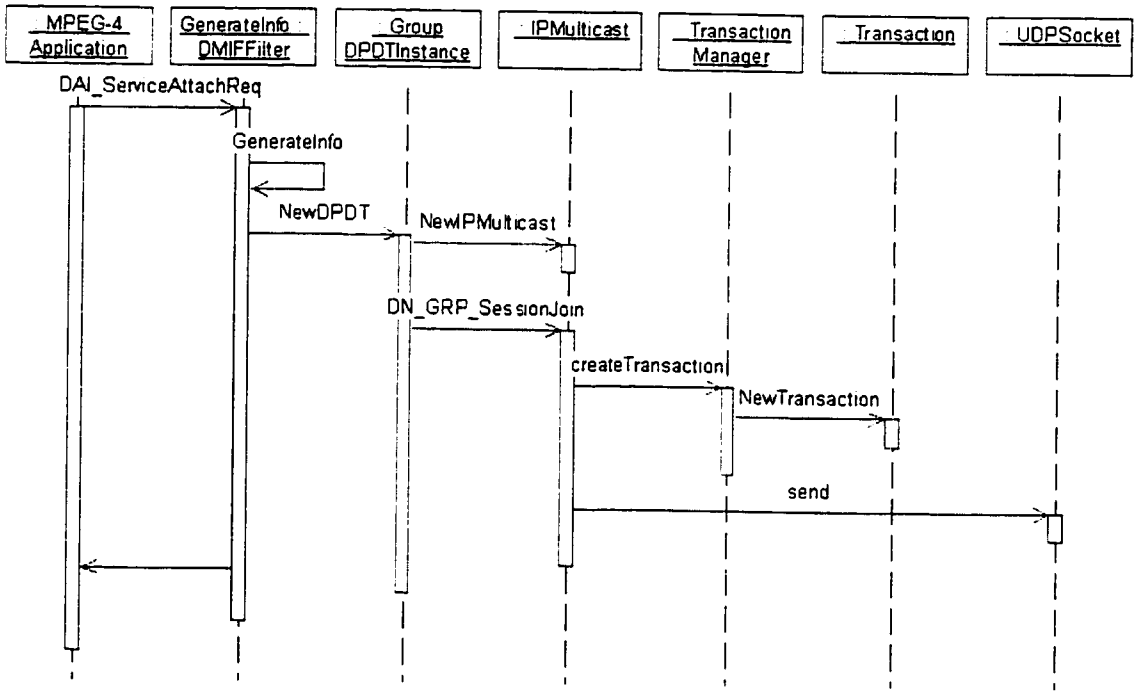

Figure 6.9 Producer service attach sequence diagram

6.5.2 Consumer Service Attach

In this scenario, the MPEG-4 Application wants to subscribe to a group session as a consumer (DPDT). Figure 6.10 shows the basic interactions among the implicated objects. Many details are not shown in this diagram, such as the creation of the transactions, which is similar to the previous scenario.

When the application wants to attach to a new group session as a consumer, it calls the DAI_ServiceAttachReq method of the DMIFFilter. The DMIFFilter parses the passed

URL to identify the required delivery technology, which is multicast transmission in this case. It creates a new service session id for the application and it creates a new GroupDCDTInstance object, too. The new GroupDCDTInstance creates a new IPMulticast object to handle the signaling protocol.

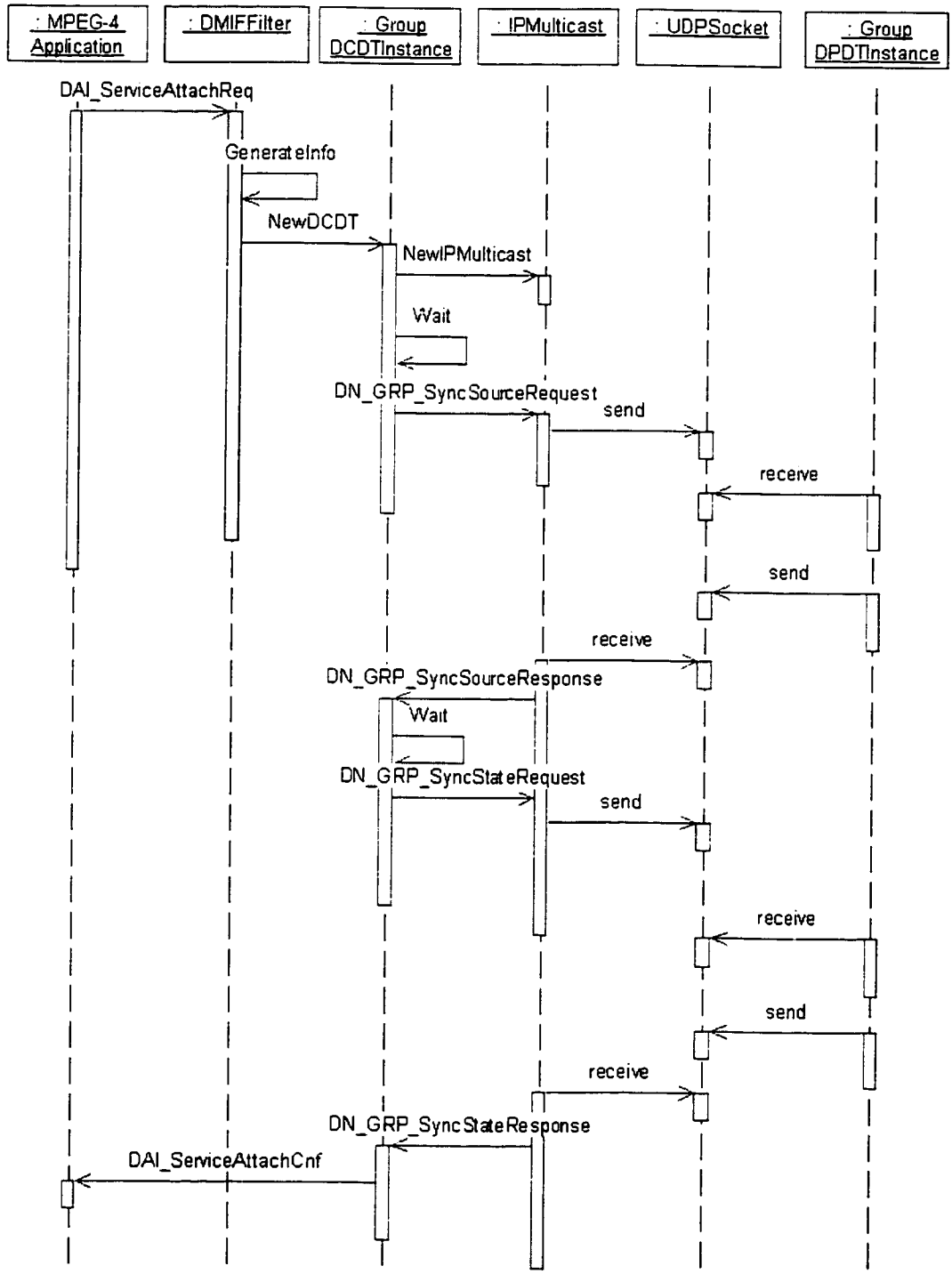

Figure 6.10 Consumer service attach sequence diagram

Then, it starts the multicast signaling protocol procedure. Following the state transition diagram of the DCDT object, it starts to record the multicast messages. Suppose that it receives no messages. After the expiration of the timer, it requests for the source ids of all the DPDTs of the group session. Every GroupDPDTInstance of the group responses to this request with the appropriate message. Then, the DCDT requests the state from either all the active DPDTs or specific ones. It waits again to collect the responses and at the end, it informs the application about the available data sources and the transport channels.

6.5.3 Channel Addition

This scenario describes the interaction among the DMIF objects to add a new transport channel. Figure 6.11 shows the corresponding UML sequence diagram.

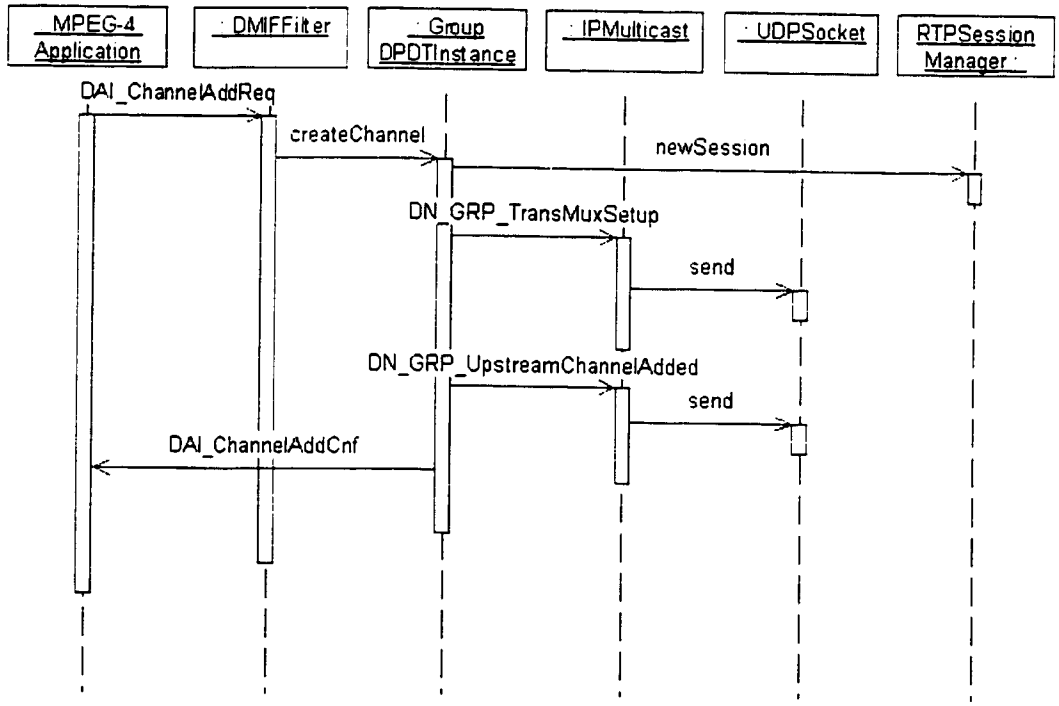

Figure 6.11 Channel addition sequence diagram

When a data producer application wants to add a new transport channel, it calls the DAI_ChannelAddReq method of the DMIFFilter. The DMIFFilter checks the used service session id and it retrieves the appropriate DMIF instance from a hash table. The chosen GroupDPDTInstance creates a new session through the RTPSessionManager for

the new channel. Then, it passes this information to a TransMuxSetup message and the IPMulticast object multicast it to all the participants of the group. This way, it informs everybody about the new physical connection. Next, it sends a new message, where it announces the addition of a new channel over the newly created connection. In the end, it confirms the addition of the new channel to the application.

6.5.4 Channel Deletion

In this scenario, a data producer application deletes an existed channel. Figure 6.12 shows the corresponding UML sequence diagram.

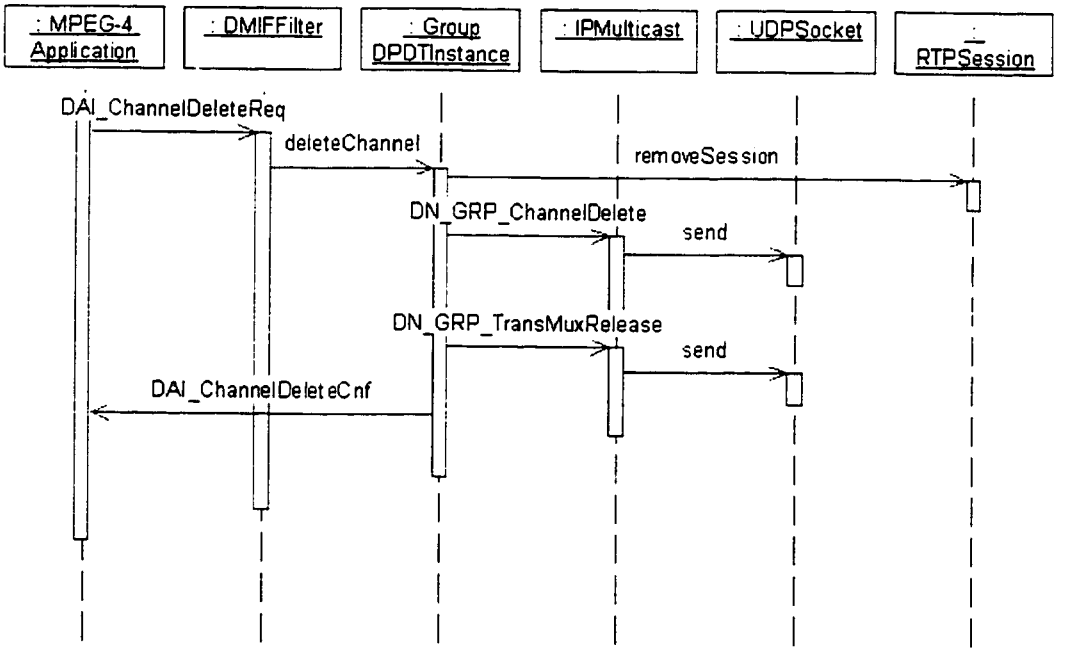

Figure 6.12 Channel deletion sequence diagram

The MPEG-4 Application calls the DAI_ChannelDeleteReq method from the DMIFFilter. The DMIFFilter picks up the appropriate DMIF instance to handle the request. The chosen GroupDPDTInstance object request the release of the session from the RTPSessionManager. Then, it sends the message channel delete to group session, to announce the deletion of that channel. Next, it sends a message to release the reserved network resources. Finally, it confirms the channel deletion to the MPEG-4 Application.

6.5.5 *Producer Service Detach*

This scenario describes the detachment of a data producer from a group session. Figure 6.13 shows the corresponding UML sequence diagram.

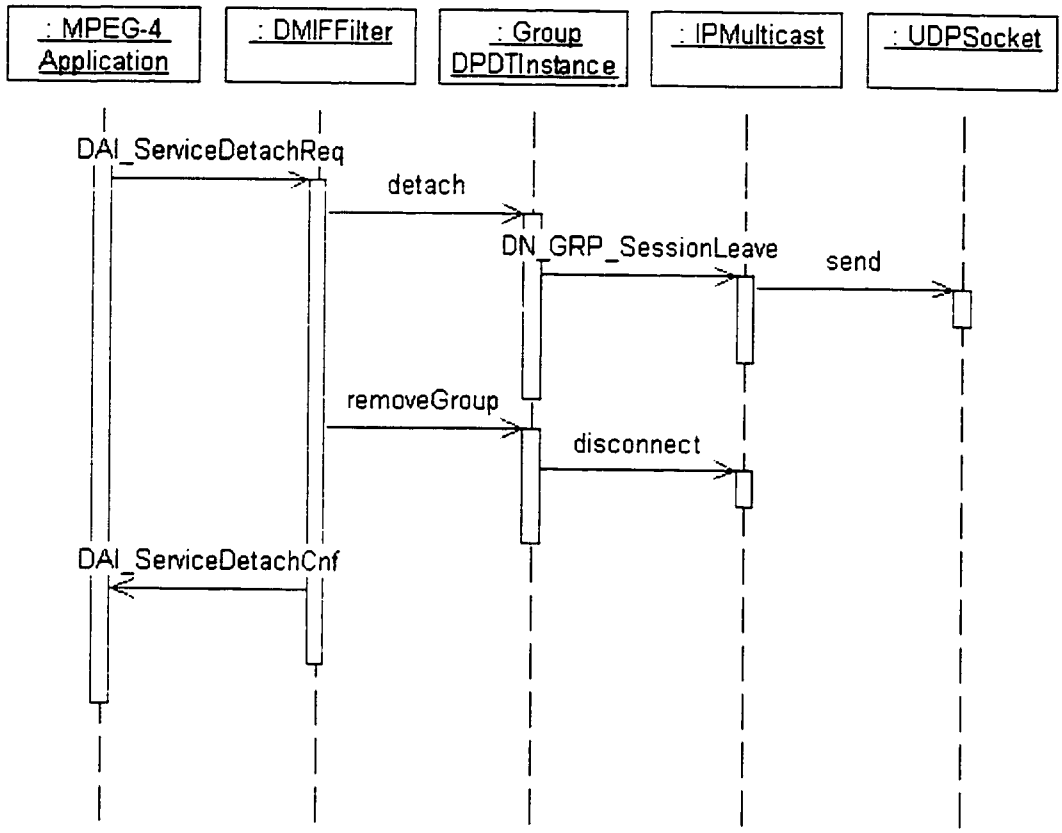

Figure 6.13 Producer service detach sequence diagram

The MPEG-4 Application calls the DAI_ServiceDetachReq from the DMIFFilter. The DMIFFilter chooses the appropriate DMIF Instance. based on the service session id. The chosen GroupDPDTInstance object calls the DN_GRPSessionLeave methods of the IPMulticast object to request the transmission of the corresponding signaling message. After, the DMIFFilter requests from the DMIF instance to remove from the group session, which in turn, requests from the IPMulticast group to disconnect from the multicast session. In the end, the DMIFFilter confirms the detachment to the MPEG-4 Application.

6.5.6 *Consumer Service Detach*

In this scenario, a data consumer application wants to detach from a group session. Figure 6.14 shows the corresponding UML sequence diagram.

The MPEG-4 Application calls the DAI_ServiceDetachReq from the DMIFFilter. The DMIFFilter retrieves the corresponding DMIF instance, based on the passed service session id. Then, it requests from the GroupDCDTInstance to remove from the group session, which in turn, requests from the IPMulticast object to disconnect from the multicast address. No messages are sending. At the end, the DMIFilter confirms the detachment to the MPEG-4 Application.

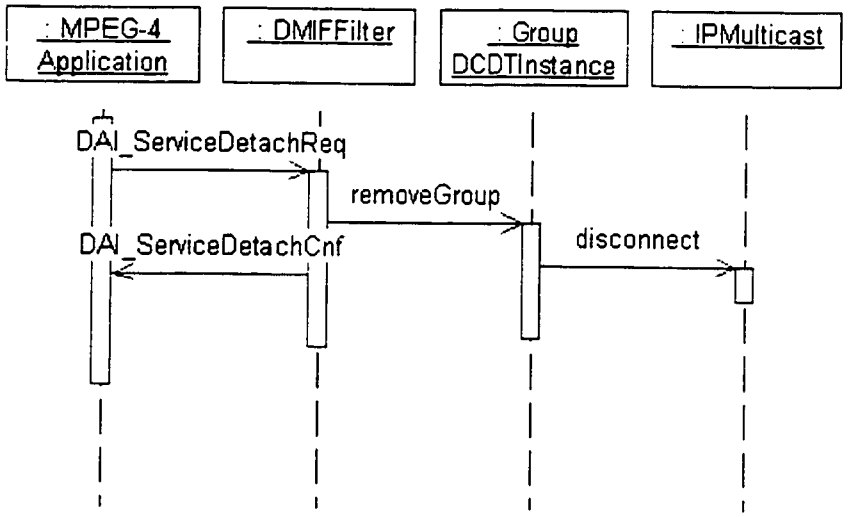

Figure 6.14 Consumer service detach sequence diagram

Chapter 7

7 Elementary Stream Management and Delivery

This chapter is focused on the management of the elementary streams at the MPEG-4 terminal and their delivery to it. Basically, two different types of streams are identified: real-time streams and streams that require reliable transmission.

Each elementary stream received in the MPEG-4 terminal, needs to be demultiplexed, decoded and composed to construct the final scene. Synchronized processing of each stream is required to enable their correct composition.

7.1 General

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.

Figure 7.1 shows the basic elementary streams into the MPEG-4 terminal architecture. 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 protocol or alternatively, with some restrictions, directly over UDP. The FlexMux tool can be used if no multiplexing is provided by the transmission protocol, or if it is very costly.

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.

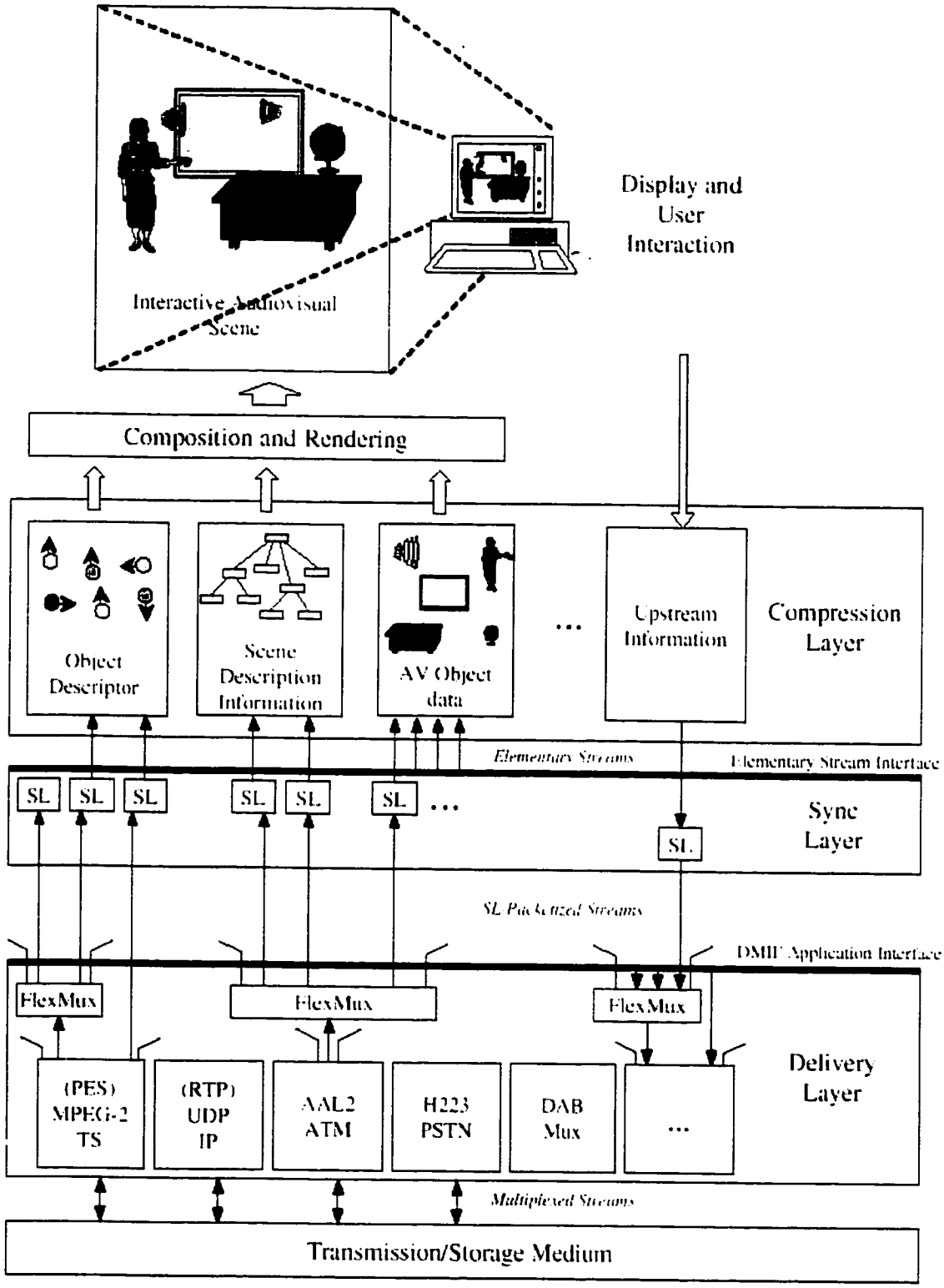

Figure 7.1 Elementary streams in the MPEG-4 terminal [29]

7.2 *Systems Decoder Model*

Systems Decoder Model (SDM) is used to specify the behavior of a receiving MPEG-4 terminal in terms of a timing model and a buffer scheme. In contrast with MPEG-1 or MPEG-2, the MPEG-4 SDM receives individual elementary streams from the delivery layer through the DMIF-Application Interface (DAI). This design has been chosen because there is no single mandatory protocol stack below the DAI that accomplishes the multiplexed delivery of elementary streams. Therefore, it is not possible to extend the model to cover multiplexing in a generic, yet meaningful way. MPEG-4 puts requirements on the end-to-end delivery of data through the DAI (constant end-to-end delay). It is considered a task for the delivery layer below the DAI to guarantee this constant delay by suitable means. As a side effect, MPEG-4 neither specifies the additional end-to-end delay induced by multiplexing nor the associated delivery jitter [24].

7.2.1 *Architecture*

The system decoder model is outlined in Figure 7.2, and consists of the DAI, a number of decoding buffers, decoders, composition memories and the compositor.

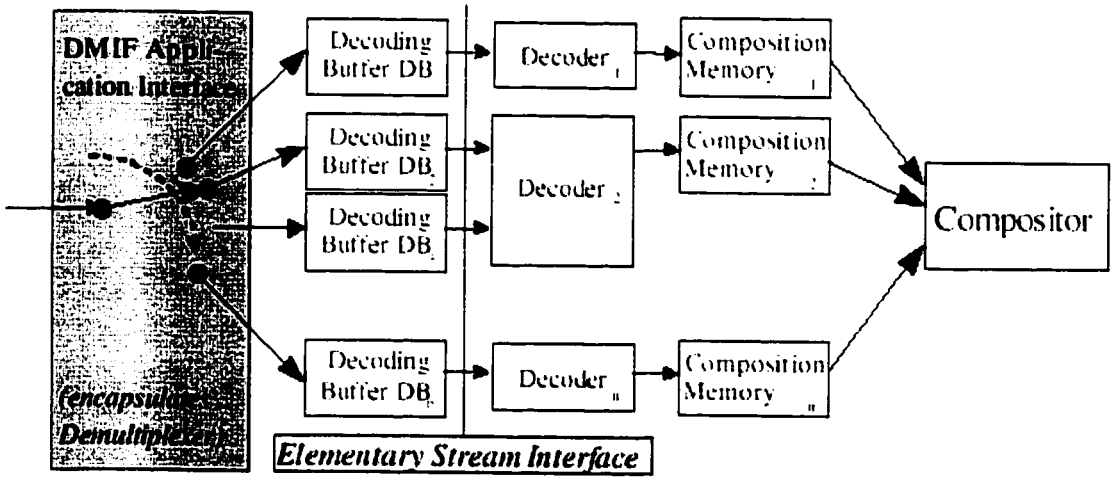

Figure 7.2 Systems Decoder Model [29]

The core entity for the purposes of the SDM is the access unit (AU). Each elementary stream is partitioned in a sequence of such AUs. For example, AU for a video stream can be each frame, while for a BIFS stream, each command.

The DAI supplies access units or parts thereof to the decoding buffer that stores the access units until their decoding time. At that point in time, SDM assumes instantaneous decoding of the access unit, removal of the access unit data from the decoding buffer, and appearance of the decoded data in the associated composition memory.

A decoder outputs the decoded data to one composition memory. The decoded data is grouped in composition units. Each composition unit of decoded data is available for composition starting at an indicated composition time, either known implicitly or through an explicit composition time stamp, and ending at the composition time of the subsequent composition unit.

7.2.2 Time model

The timing model postulates Object Time Bases (OTB) to which the timing of elementary streams shall adhere. Such a time base is actually established at the sending side. Since in general it cannot be assumed that the sender and receiver are locked to the same clock, samples of the time base can be communicated to the receiver by means of Object Clock Reference (OCR) time stamps. The receiver can then estimate, and therefore reconstruct, the speed of the sender clock by observing the arrival times of such OCRs.

In general, audio-visual objects whose elementary streams have different OTBs may be combined into a single presentation. A typical case would be a multi-point collaborative system, where it is not required that all sources run on the same time base. It is therefore not easily possible to synchronize contributions from different locations with each other in a strict sense, since there is no common, absolute, clock. Therefore, the terminal has to recover the time bases from all contributing sources in order to present associated audio and video composition units at the right point in time [24].

7.3 Media Delivery with JMF RTP

As it has already been mentioned at Chapter 2, Java Media Framework (JMF) provides an API to transmit media over the RTP protocol.

7.3.1 *Real-time Transmission*

As Real-time transmission is considered the transmission of real-time streams, where every Access Unit has to be delivered with a specific maximum delay. RTP is a real time transmission protocol and the required functionality is provided by the implementation of JMF RTP.

In JMF, the SessionManager interface is used to coordinate an RTP session. The SessionManager keeps track of the session participants and the streams that are being transmitted. Also, it maintains the state of the session as viewed from the local participant. In effect, the SessionManager is a local representation of a distributed entity, the RTP session. In addition, it handles the *RTCP* control channel, and supports RTCP for both senders and receivers. The SessionManager interface defines methods that enable an application to initialize and start participating in a session, remove individual streams created by the application, and close the entire session.

The SessionManager maintains a RTPStream object for each stream of RTP data packets in the session. There are two types of RTP streams:

- ReceiveStream, which represents a stream that is being received from a remote participant.
- SendStream, which represents a stream of data coming from the Processor or input DataSource that is being sent over the network.

A ReceiveStream is constructed automatically whenever the session manager detects a new source of RTP data. SendStreams are created after a specific request. Each RTP stream has a buffer data source associated with it. For ReceiveStreams, this DataSource is always a PushBufferDataSource.

The abstract class RTPPushDataSource defines the basic elements of a JMF RTP data handler. A data handler has both an input data stream (PushSourceStream) and an output data stream (OuputDataStream). A data handler can be used for either the data channel or the control channel of an RTP session. If it is used for the data channel, the data handler implements the DataChannel interface.

7.3.2 *Reliable Transmission*

Reliable transmission is a transmission that ensures the correct delivery of a packet at the destination, even in error-prone environments. Reliable transmission requirements are controversial to the requirements of real-time transmission. Reliable transmission is usually implemented by re-transmitting the lost packets, thus, increasing unpredictably the overall delay. This is not acceptable for real-time transmission of media.

JMF RTP is extended to become a reliable protocol. The implementation is very similar to the Lightweight Reliable Multicast Protocol (LRMP) [40]. RTCP App packets are used to report lost packets.

More specifically, each receiver of the multicast group checks the counter of each RTP packet. If the lost of a packet is detected, the receiver sends a Negative Acknowledgement (NACK) to the source, requesting for retransmission of the packet. If it receives other packets (which are already on the network) it stores them at a cache memory, to process them, after the reception of the lost packet. If it receives duplicated packets, which are requested by other receivers, it drops them.

On the other hand, a sender keeps the transmitted packets at another cache memory to be able to retransmit them, in case it is requested to. The size of this cache memory is based on the Round Trip Time (RTT) of the particular session.

This mechanism works efficiently, if the rate of lost packets is relatively small. A more advanced mechanism is required for networks with high error rates.

7.4 *BIFS over JMF RTP*

JMF RTP has been designed for the transmission of audio and video streams, mainly. But, it is possible to be extended to transmit other media. Although, the process is not specifically determined, the system provides an implementation of this extension, which enables the transmission of MPEG-4 BIFS over the JMF RTP implementation. In addition, JMF is extended to include MPEG-4 BIFS encoder/decoder and packetizer/ depacketizer.

A new BifsStream class is defined to model the BIFS streams. An implementation of BIFS-related DataSource provides the BifsStream to Processors, Players or DataSinks. This DataSource is related to a new implementation of Processor that handles BIFS

streams. The mapping of the BIFS streams on RTP packets is manipulated by a BIFS-based packetizer/depacketizer.

Finally, a new payload format has been defined in this thesis to describe the MPEG-4 BIFS streams. It is considered as a Video Stream, although it is not. This assumption has been made to make possible the implementation as an extension of JMF.

JMF Registry is updated to include the implementations of the new components, both for protocols/services (reliable RTP) and content (MPEG-4 BIFS).

Chapter 8

8 Application Development Issues

In this chapter, an example application, the MPEG-4 Browser, is presented. It is a prototype application, which uses the developed system to achieve collaboration among users, in a 3D environment, integrating media streams into it.

Every application that uses that system has to follow some requirements. Based on those requirements, the architecture of the main components is designed. The rendering mechanism is very important to enable the visualization of the MPEG-4 content. In addition, the application should be DMIF-compliant. Finally, the implemented Graphical User Interface is shown.

8.1 Architecture Design

The high level architecture of the MPEG-4 Browser is shown in Figure 8.1. The identified components are essential to every application that uses this framework.

The DMIFApplication is an interface that provides the callback functions for the DMIF subsystem. This way, an application can make a request to the DMIF subsystem and then continue its operation flow, without to wait for the response from the framework. When the response is ready, DMIF informs the application, using the DMIFApplication interface.

Users can interact with the application through a Graphical User Interface (GUI) that provides ways to open VRML files, connect to multicast groups to join to collaborations and navigate into the loaded virtual environments.

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 a 3D browser and the Video renderer, which renders the decoded video frames onto the surfaces of 3D objects.

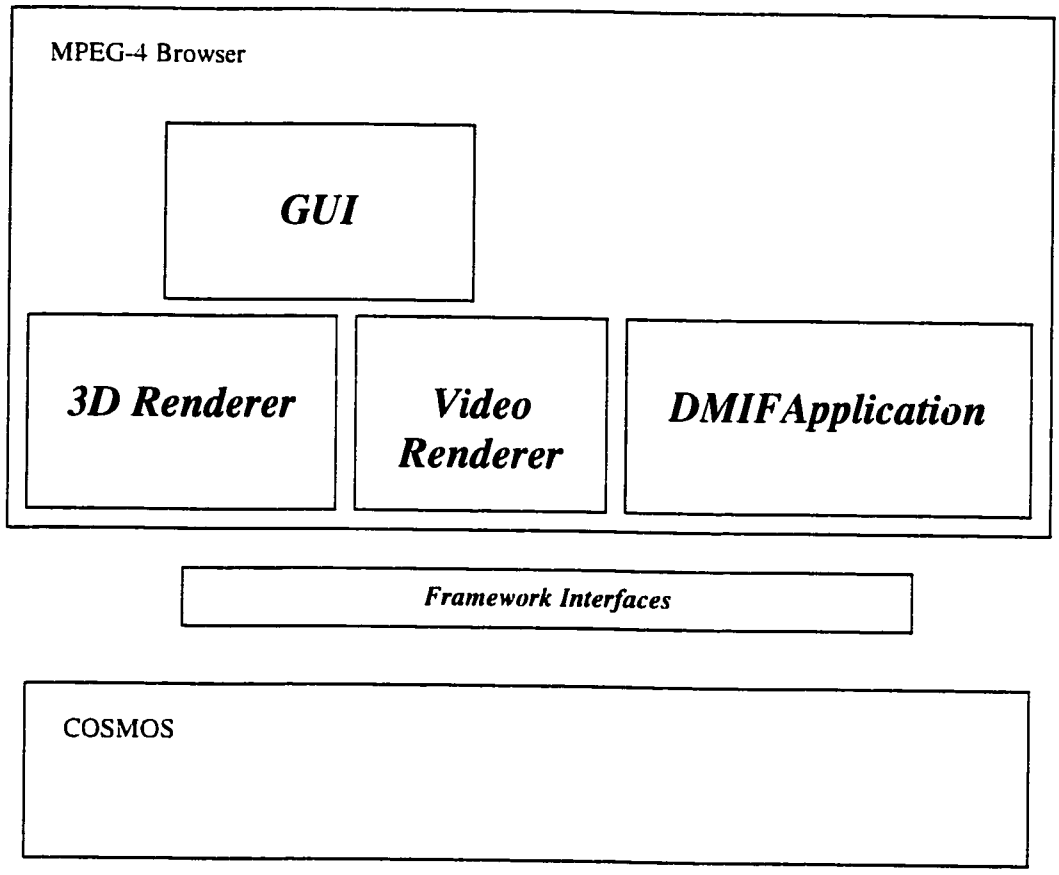

Figure 8.1 Application Architecture

8.2 Rendering Mechanism

As it has already been mentioned, the rendering of the visual content is handled by two components: the 3D Renderer and the Video Renderer. Aural content is handled by JMF decoders and renderers, but they are lacking the capability of 3D sound.

8.2.1 3D Rendering

8.2.1.1 Rendering mode

A basic introduction to Java 3D has been given in Chapter 2. Java 3D includes three different rendering modes: immediate mode, retained mode, and compiled-retained mode. Each successive rendering mode allows Java 3D more freedom in optimizing an

application's execution. In the implementation of this application the compiled-retained mode has been used.

Compiled-retained mode requires the application to construct a scene graph and specify which elements of the scene graph may change during rendering. Additionally, the application can compile some or all of the subgraphs that make up a complete scene graph. Java 3D compiles these graphs into an internal format. The compiled representation of the scene graph may bear little resemblance to the original tree structure provided by the application, however, it is functionally equivalent. Compiled-retained mode provides the highest performance.

8.2.1.2 Canvas3D

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 extends the java.awt.Canvas object to include 3D-related information such as the size of the canvas in pixels, the Canvas3D's location, also in pixels, within a Screen3D object, and whether or not the canvas has stereo view enabled.

The Canvas3D object is connected to the Universe object, which is the top object in the hierarchy of a Java 3D implemented environment. This way, every object of the Universe is rendered into the Canvas3D.

8.2.1.3 Node Mapping

An MPEG-4 node is a structure that keeps information regarding the values of the fields. It does not provide visual information 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. Appendix C gives the hierarchies of those objects.

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.

Another important example is the TouchSensor node. It is a behavior node, which generates messages, when the user interacts (with the mouse) with specific objects into the 3D scene. TouchSensor encapsulates an implementation object, which is called JMTouchSensor. JMTouchSensor extends JMPickMouseBehavior class and implements the MouseBehaviorCallback interface to handle the interactions of the user. JMPickMouseBehavior extends the Behavior class, provided by Java3D to handle behavior actions in general.

The ability to navigate into the world has been developed in a similar way.

8.2.2 Video Rendering

JMF provides a set of encoders/decoders to encode/decode video and audio streams, in the most of the 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.

But, MPEG-4 requires rendering mechanisms that are capable of providing computer-generated material with natural-origin captured media streams. For this reason, new rendering mechanisms have been implemented, by extending the functionality of JMF to render the video streams onto the surfaces of 3D objects.

In reality, each video frame is decoded, and then is stored in special buffers of the JMFImageTexture class. JMFImageTexture is encapsulated within the MPEG-4 ImageTexture node. ImageTexture node is a field of the Appearance node, which in turn is a field of the Shape node. Periodical setting of the JMFImageTexture buffer enables the rendering of each frame.

JMFImageTexture implements the javax.media.renderer.VideoRenderer interface, and it is inserted as a PlugIn into the JMF implementation. The right scheduler object periodically provides the renderer with decoded video frames. The scheduler is responsible for both the intra- and inter-stream synchronization.

The analysis of the displayed frames can be set parametrically to optimize the ratio of the image quality with the displayed frame rate.

8.3 DMIF Compliance

As it has already mentioned, every application that uses the DMIF subsystem should implement the DMIFApplication interface. The DMIFApplication interface provides a set of methods that enables the DMIF implementation to confirm a request or to indicate a new update at the state of the group session.

For example, when a new Consumer joins a group session, it requests a detailed list of the available services and data channels. After the request, the control is returned to the application t enable further interactions with the user. The implementation of the DMIF is thread-based. Thus, in parallel, it collects all the requested information and when that procedure finishes, it confirms the request and provides the requested data. Similarly, when a new Producer joins the group session, each receiver is informed of the new service and data channels. Then, the DMIF implementation indicates the new data to the application, through the callback interface. Users can then add the new streams in the application, as long as they are interested in them.

In addition, when there is a QoS violation, the DMIF implementation uses the callback interface to inform the application, which might decide a degradation of the requested QoS.

8.4 Graphical User Interface

Figure 8.2 shows a screen-shot of the MPEG-4 Browser application. The biggest part of the window is covered with the Canvas3D. There, the contents of the world are displayed.

In the shown screen-shot, a VRML world has been loaded. It is comprised of an office scene, which includes a desk, a chair, a computer desk, and a monitor. An image has been loaded in the background, which contains a view of a city. Also, a rectangular box is into the office that is used for the rendering of a video stream. More, an avatar of another user is also into the scene. This user shares the same virtual environment.

The control component of the application provides a set of basic actions. A user can load a VRML or BIFS file, or store a scene in BIFS format to disk. He can also have a second view of the same scene. This feature enables the concurrent display of objects form different views.

In addition, DMIF functionality is available. A User can join a group both as a Consumer or a Producer. If he joins as Producer, he can transmit the content of his scene to the other participants of the group.

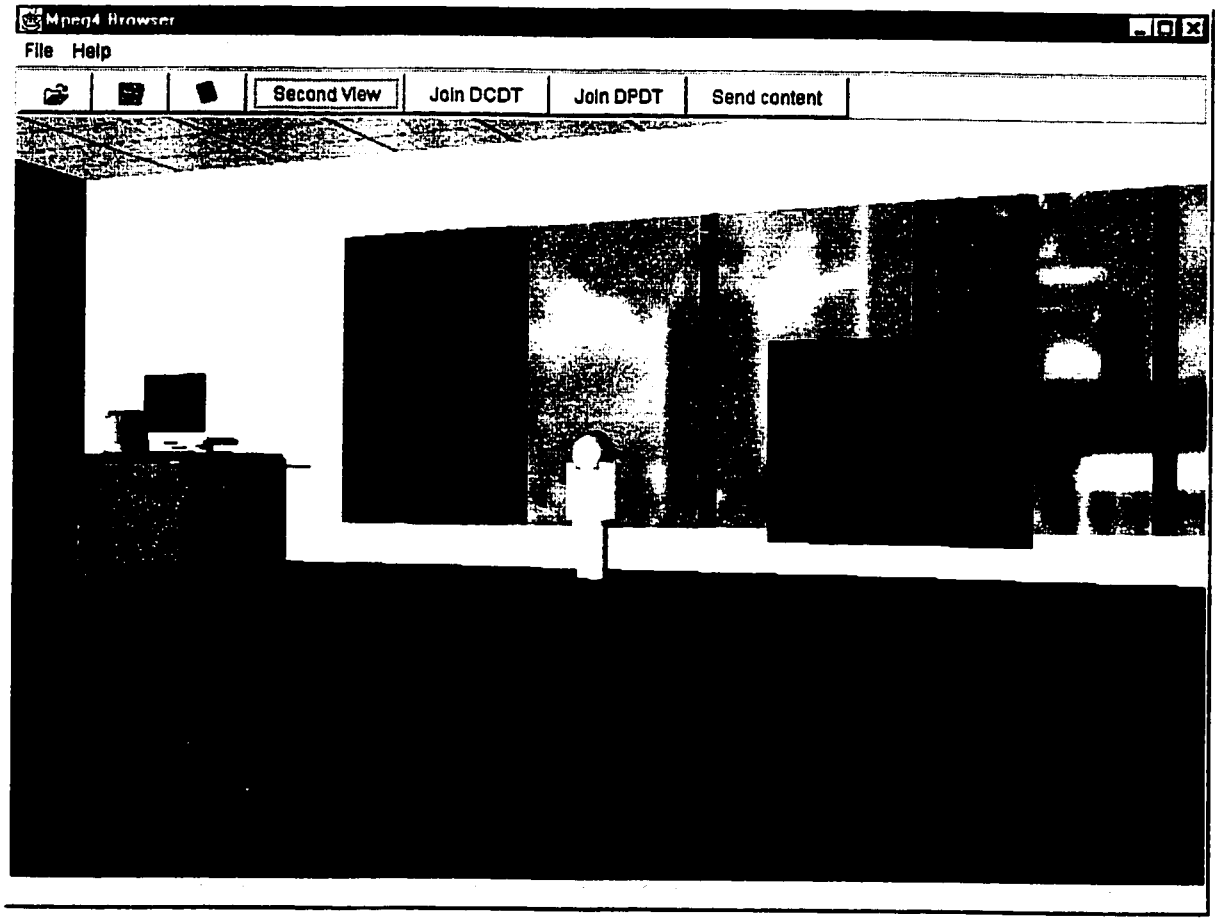

Figure 8.2 MPEG-4 Browser screen-shot

Figure 8.3 shows the window that is presented to a user who wants to join a multicast group, as a Producer. The URL of the group is specified and is passed at the DMIF layer, when the user presses the Join button. He can add channels, which are advertised every DCDT participant of the group. Also, he can remove existed channels, or leave the group session.

A similar window is presented for Consumers.

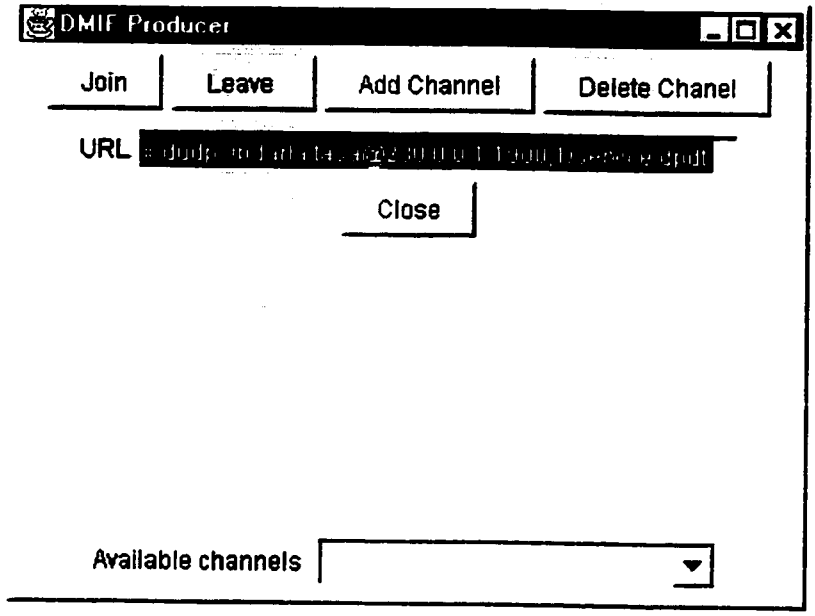

Figure 8.3 DMIF Pop-up window

8.5 Performance Evaluation

MPEG-4 Browser is capable of loading VRML or MPEG-4 BIFS files and rendering them into the Canvas3D frame. Loading and scene constructing time is relatively long, compared to other existing commercial Browsers. For example, it requires almost double time to load and render a large scene, compared to the CosmoPlayer browser, which is capable of loading VRML files only. The bottleneck of the loading task is the construction of the Java 3D objects. Of course, MPEG-4 Browser offers much more advanced characteristics than CosmoPlayer.

The video display frame-rate of the MPEG-4 Browser is slightly slower than the frame-rate of a normal video application, which renders each frame in a rectangular window. Again, Java 3D is responsible for this difference. The achieved frame-rate of the MPEG-4 Browser is something between 10 to 15 frames per second, for QCIF size frames, and good display quality. Of course, the encoding format of the video influences the resulted rate. AVI or Quick Time encoded videos have better performance than MPEG-1 ones.

Navigation into the 3D worlds is almost similar to that of CosmoPlayer. This is achievable, by using optimized rendering techniques.

Chapter 9

9 Conclusions and future extensions

The last chapter of the thesis provides the conclusions and summarizes the contribution of the system. In addition, the most important of the possible future extensions of the system are discussed.

9.1 Conclusions

The development of a collaborative system is a very challenging task. The inherit complexity due to the fact that it combines many research areas, leads to the solution of a framework-based architectural structure, to increase the reusability of the components. Also, software design patters are necessary to provide a more efficient and easier-to-develop solution. UML can provide the means for describing the design of such systems, although it lacks some features to real-time execution characteristics and layer-based design.

MPEG-4 does not specify the way a collaborative system should be developed. But it provides a set of tools to handle many issues that arise during the development of a collaborative system, which needs a way to describe 3D virtual environments and stream them among the participants. In addition, it specifies a delivery framework for multicast group session establishment and release.

VRML is a very popular format for the description of models of 3D virtual environments, thus it can not be ignored in the design of the system. But it is not designed to stream scene descriptions over networks. MPEG-4 provides an alternative solution, the BIFS format, which compresses the scene description information and enables the streaming over networks.

Moreover, the integration of multimedia streams, such as video and audio, with computer-generated graphics in the same rendering frame, provides an extremely

powerful way to describe scenes. Applications can decide the tradeoff between the parametrically specified computer-generated graphics and the high bandwidth consuming media streams.

Multicast delivery mechanisms reduce significantly the load of the network traffic, which is generated from the collaborative system. Although it is much harder to provide reliable transmission compared to a unicast solution and it is not available for Wide Area Networks yet, multicast delivery is the preferred solution. RTP is the proposed transport protocol for the transmission of multimedia over the Internet.

Interaction of the user with the content is also a necessary characteristic of a collaborative system. An event-based architecture can efficiently handle user interactions, using a thread for each event.

Finally, Java is a very powerful language that enables fast development of complicated systems. The language is consisted of a core part with the basic functionality and a set of extensions to handle special requirements. JMF is one of them and it provides a framework for multimedia processing and transmission. In addition, Java 3D API is very helpful for the development of 3D applications. Also, JavaCC can be used to develop text-based parsers. It has been used for the development of the VRML parser.

9.2 Summary of contribution

The following list summarizes the contribution of this thesis:

- Design and implementation of the MPEG-4 DMIF part, for multicast scenarios.
- Design and implementation of an MPEG-4 BIFS encoder/decoder tool.
- Design and implementation of a framework, which describes 3D scenes with a directed acyclic graph.
- Design and implementation of a rendering mechanism for 3D scenes integrated with the rendering of video streams.
- Design and implementation of a VRML parser to "read" VRML files and their conversion to BIFS streams.
- Design and implementation of a streaming framework to deliver the media to each participant of a collaborative group.
- Design and implementation of a subsystem for the interaction with the user.

- Design and implementation of an application that demonstrates the usage of the basic functionality of the framework.

9.3 Future research

The implemented system provides a basic functionality for collaborative systems. The design of the system is highly extendable and modular, and allows easy integration with new features, which will improve the developed system. Since, MPEG-4 Version 2 is still under development, this system does not include the new - even more attractive - characteristics of the last version of the standard.

The following list provides the most important, possible extensions of the system:

- Implementation of every MPEG-4 node. The current version supports only a subset of the nodes that are defined in the MPEG-4 standard. The chosen nodes are the most important, but the rest can give a more powerful way to describe scenes.
- MPEG-4 Version 2 Systems includes the definition of an API, which is called MPEG-J. It provides a Java interface and architecture, through which applications access basic object of the framework using a well-defined interface. It is not yet in its final form.
- MPEG-4 Systems part provides one more stream to augment the scene description: The Object Descriptors (OD) stream. This stream links the BIFS stream with other multimedia streams and provides a more powerful way to describe the scenes.
- MPEG-4 provides a set of nodes that offer 3D sound capabilities. Their inclusion in the implemented system will increase the quality of the simulated virtual environments.
- MPEG-4 DMIF API provides a set of methods to request specific QoS requirements for each stream. In addition, it offers a way to the DMIF implementation to inform the applications, when there is a QoS violation. This is a very important extension that can be combined with the RSVP protocol to provide QoS guarantees.
- RTSP is a protocol that provides VCR-like functionality for applications over the Internet. A future version of JMF will include an implementation of the RTSP protocol. The integration of the RTSP protocol with the implemented system can

improve the network usage, since it will offer a solution to stop the reception of streams, which are spatially away from the view of the user [14].

- The implemented system provides only 3D environments, integrated with the rendering of media streams. In addition, MPEG-4 provides a set of nodes for the description of 2D scenes, which are very useful for many kinds of applications that do not require 3D complexity.
- MPEG-4 specifies the requirements and the architecture of advanced video decoders, which are capable of encoding video objects of arbitrary shape. The integration of such encoders will make even more natural rendering of virtual worlds.
- MPEG-4 DMIF provides a set of specifications for remote interactive scenarios. Extension of the DMIF implementation to include them will enable the usage of the system at different application types, which are based on the client-server model.
- BIFS implementation does not provide the best possible compression. Other features, like quantization can improve significantly the compression factor.
- Finally, the system can be modified to support applets instead of applications. This way, it will be easily integrated with web applications.

References:

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

[2] O. Avaro, P. Chou, A. Eleftheriadis, C. Herpel, C. Reader, J. Signes, "The MPEG-4 Systems and Description Languages: A Way Ahead in Audio Visual Information Representation", Image Communications Journal, 1997

[3] V. Balabanian, " The Role of DMIF in Support of RTP MPEG-4 Payloads", draft-balabanian-rtp-mpeg4-dmif-01, http://ring.jah.ne.jp/pub/doc/internet-drafts/, August 1998.

[4] V. Balabanian, 'The Use of MPEG-4/DMIF and RSVP with Integrated Services' draft-balabanian-intserv-dmif. Internet Engineering Task Force, http://ring.jah.ne.jp/pub/doc/internet-drafts/ , August 1998.

[5] V. Balabanian, "The Role of DMIF with RTSP and MPEG-4", draft-balabanian-rtsp-mpeg4-dmif-00, http://ring.jah.ne.jp/pub/doc/internet-drafts/, Sep. 1998

[6] V. Balabanian, L. Casey, N. Greene, Digital Storage of Media-Command and Control Protocol Applied to ATM, IEEE Journal of Selected Areas of Communications, Vol. 14, No. 6, Aug. 1996.

[7] R. Becchini, G. De Petris, M. Gugliemo, A. Morvan, "SOMMIT Project: Enabling new services for the next generation of digital TV receivers", ECMAST 99, 1999

[8] G. Booch, "The Unified Modeling Language User Guide", Addison Wesley, 1998

[9] E. Caglar, M. Hardt, C. Hoene, S. Zander, "RTP/RTC Design and Implementation with Channels", Technische Universität Berlin, http://tragicomix.ee.tu-berlin.de/~chan03/, 1998

[10] R. Carey, G. Bel, "The Annotated VRML97 Reference Manual", http://www wasabisoft.com/Book/Book.html", 1997

[11] L. Chiariglione, "MPEG-4, why use it?", Position Paper, http:// www.cselt.it/ufv/leonardo/paper/mpeg-4/mpeg-4.htm, 1998

[12] L. Chiariglione, "The MPEG-4 standard", Journal of the China Institute of Communications, http://www.cselt.it/ufv/leonardo/paper/china98/china98.htm, September 1998

[13] P. Christ, C. Guillemot, S. Wesner, "RTSP-based Stream Control in MPEG-4", Internet Engineering Task Force, INTERNET-DRAFT, http://ring.jah.ne.jp/pub/doc/internet-drafts/, November 1998

[14] M. R. Civanlar, V. Balabanian, A. Basso, S. Casner, C. Herpel, C. Perkins, "RTP Payload Format for MPEG-4 Streams", draft-ietf-avt-rtp-mpeg-4-01.txt, Internet Engineering Task Force, http://ring.jah.ne.jp/pub/doc/internet-drafts/, February 1999

[15] V. Darlagiannis, Design and Implementation of the Stream, Session and Download Service Elements in the KYDONIA multimedia server, compliant to the DAVIC standard, Technical University of Crete, October, 1997.

[16] Ir. Defee, jami Kangasoja, M. Rustari, "COMIQS System for commercial presentations on the Internet", ECMAST 99, 1999

[17] J. Deicke, U. Mayer, A. Knoll, M. Glesner, "Flexible Multiplexing in MPEG-4 Systems", IDMS '98, 1998

[18] H. Dommel, J. Garcia-Luna-Aceves, "Group Coordination Support for Synchronous Interner Collaboration", IEEE Internet Computing, March/April 1999

[19] G. Franceschini, "The Delivery Layer in MPEG-4", Image Communication Journal, August 1999

[20] Gamma et al., "Design patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1995

[21] P. Gerken, S. Schultz, G. Knabe, F. Casalino, G. Di Cagno, M. Quaglia, J. Dufourd, S. Boughoufalah, F. Bouilhaguet, M. Stepping, T. Bonse, U. Mayer, J. Deicke, M. Glesner, "MPEG-4 PC – Authoring and Playing of MPEG-4 content for local and broadcast applications", ECMAST 99, 1999

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

[23] C. Herpel, V. Balabanian, A. Basso, M. R. Civanlar, D. Hoffman, M. F. Speer, H. Schulzrinne, "RTP payload format for MPEG-4 Elementary Streams" ietf-avt-rtp-mpeg4-dmdraft-ietf-avt-new-00, Internet Engineering Task Force, http://ring.jah.ne.jp/ pub/doc/internet-drafts/, March 1998.

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

[25] Ar. van Hoff, "The case for Java as a Programming Language", IEEE Internet Computing, Januaty/February 1997

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

[27] ISO/IEC 13818-1, "Information Technology - Generic Coding of Moving Pictures and Associated Audio: Systems", ISO/IEC JTC1/SC29/WG11 N0801rev, Apr. 1995.

[28] ISO/IEC 13818-6, "Information Technology - Generic Coding of Moving Pictures and Associated Audio: Digital Storage Media Command and Control International Standard", ISO/IEC JTC1/SC29/WG11 MPEG96/N1300p1, July, 1996.

[29] 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

[30] 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

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

[32] 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

[33] ISO/IEC 14772-2:1997, "Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML) -- Part 2: External authoring interface."

[34] ISO/IEC JTC1/SC29/WG11, "MPEG Systems (1-2-4-7) FAQ, Version 6.0", http://drogo.cselt.stet.it/mpeg/faq/faq-systems.htm , October 1998

[35] ISO/IEC JTC1/SC29/WG11, "Overview of the MPEG-4 Standard", http://drogo.cselt.stet.it/mpeg/faq/faq-systems.htm , July 1998

[36] R. Koenen, "Overview of the MPEG-4 Standard", http://drogo.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm, July 1998

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

[38] R.Koenen, F.Pereira, L.Chiariglione, "MPEG-4: context and objectives", Image Communication Journal, vol. 9, n° 4, May 1997, http://www..cselt.it/ufv/leonardo/icjfiles/mpeg-4_si/paper1.htm

[39] X. Li, M. Ammar, S. Paul, "Video Multicast over the Internet", IEEE Network, March/April 1999

[40] T. Liao, "Light-weight Reliable Multicast Protocol as an Extension to RTP", 1997

[41] C. Marrin, "Beyond VRML", http://www.marrin.com/vrml/private/ EmmaWhitePaper.htm , 1998

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

[43] S. Meloan, "Java Media Framework 2.0", JMF Article, http://java.sun.com/features/1999/09/multimedia.html , September 1999

[44] C. Metz, "Reliable Multicast: When many must absolutely positively receive it", IEEE Internet Computing, July/August 1998

[45] D. Nadeau, "Building Virtual Worlds with VRML", IEEE Computer Graphics and Applications", March 1999

[46] H. Ohzu, K. Habara, "Behind the Scenes of Virtual Reality, Vision and Motion", Proceedings of the IEEE, Vol. 84, No. 5, May 1996

[47] J. Ostenmann, "MPEG-4 Overview", Circuits and systems in the information age, June 1997

[48] F. Pereira, "MPEG-4: Why, What, How and When?", Image Communication Journal, August 1999

[49] A. Puri, A. Eleftheriadis, "MPEG-4: An Object-based Multimedia Coding Standard supporting Mobile Applications", ACM Mobile Networks and Application Journal, Special Issue on Mobile Multimedia Communications, http://www.ee.columbia.edu/~eleft/publications.html, August 1997

[50] Roehl B., "Draft Proposal for the VRML Streaming WG", http://ece.uwaterloo.ca/~broehl/streams/proposal.html , June 1998

[51] H. Schulzrinne, Lanphier "Real Time Streaming Protocol (RTSP)" RFC 2326, Internet Engineering Task Force, ftp://ftp.is.co.za/rfc/, April 1998

[52] 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

[53] J. Signes, Y. Fisher, A. Eleftheriadis, "MPEG-4: Scene Representation and Interactivity", Multimedia Systems, Standard, Networks, 1999 (to appear)

[54] J. Signès, Y. Fisher, A. Eleftheriadis, "MPEG-4's Binary Format for Scene Description", Image Communication Journal, August 1999

[55] H. Sowizral, M. Deering, "The Java 3D API and Virtual Reality", IEEE Computer Graphics and Applications, May/June 1999

[56] R. Steinmetz, K. Nahrstedt, "Multimedia: Computing, Communications and Applications", Prentice Hall, 1995

[57] SUN Microsystems, Inc., "Java Media Framework API guide", September 1999

[58] SUN Microsystems, Inc., "JMF API Data Sheet", http://java.sun.com/marketing/collateral/jmf_ds.html,1998

[59] SUN Microsystems, Inc., "The Java Language"

[60] SUN Microsystems, Inc., "Java 3D API Guide Specification", December 1998

[61] VRML97, ISO/IEC 14772-1:1997 – History http://www.vrml.org/WorkingGroups/vrml-eai/history.html

[62] VRML-MPEG4 WG, "Main differences between MPEG-4 and VRML97", http://www.vrml.org/WorkingGroups/vrml-mpeg4/differences.html , 1998

[63] Mailing lists for Java3D, JMF, MPEG-4/VRML and DMIF development group

[64] SUN Microsystems, Inc., "JavaCC - The Java Parser Generator", http://www.metamata.com/JavaCC/

Appendix A

A. UML Diagrams

This appendix discusses the notation of four UML diagrams, which are used in this document:

- Use case diagrams
- Class diagrams
- Sequence diagrams
- Collaboration diagrams

A Use case diagram presents the usage scenarios of a system, in a high level way. Figure A.1 shows an example diagram.

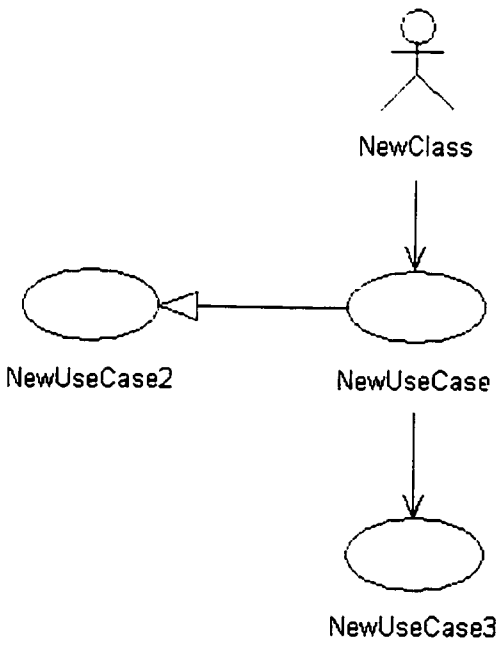

Figure A.1 UML Use Case example diagram

Actors are represented with human-like figures. Use cases of the system are represented with ovals. An actor is connected with a use case, using a line with arrow (in the example, NewClass is connected with NewUseCase). A use case can extend another use case (in the example, NewUseCase extends NewUseCase2), or use another use case (in the example, NewUseCase uses NewUseCase3).

A UML class diagram shows the relationships among the classes and the interfaces of the system. Figure A.2 shows an example of UML class diagram. The NewClass2 class implements NewInterface interface. It is denoted with a dashed line with close arrow. NewClass2 aggregates NewClass3. This relationship is denoted with a continuous line with a diamond. NewClass2 extends NewClass4. This relationship is denoted with a continuous line with a closed arrow. NewClass2 is associated with NewClass5, and this relationship is denoted with a continuous line. Finally, NewClass4 realizes NewClass6.

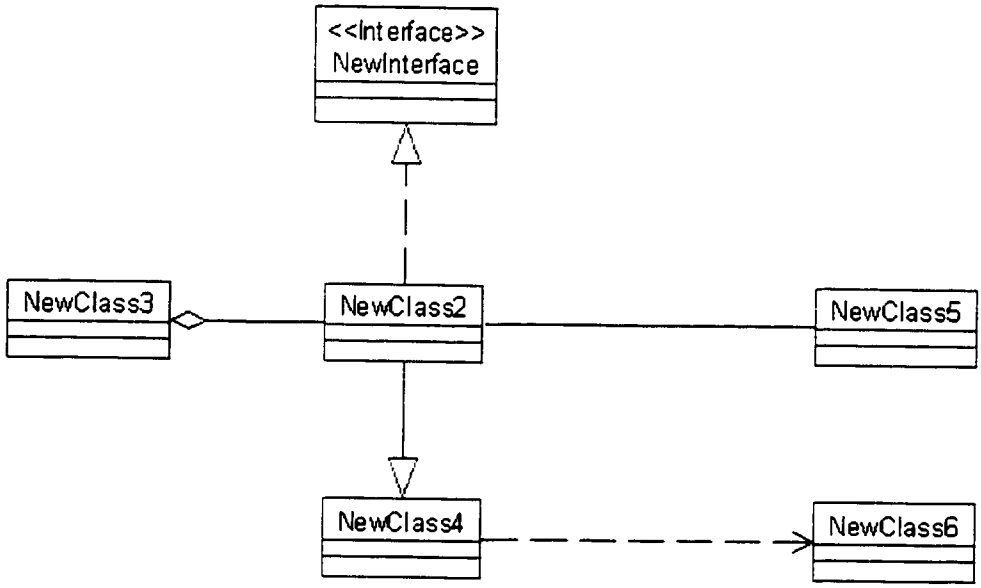

Figure A.2 UML class example diagram

UML sequence diagrams shows the way the objects of the system exchange messages with each other. Figure A.3 shows an example diagram. NewClass2 calls the "a" method of NewClass3, which calls the "b" method" of NewClass4. Then, NewClass3 calls the "c" method of NewClass2, which calls the "d" method of itself.

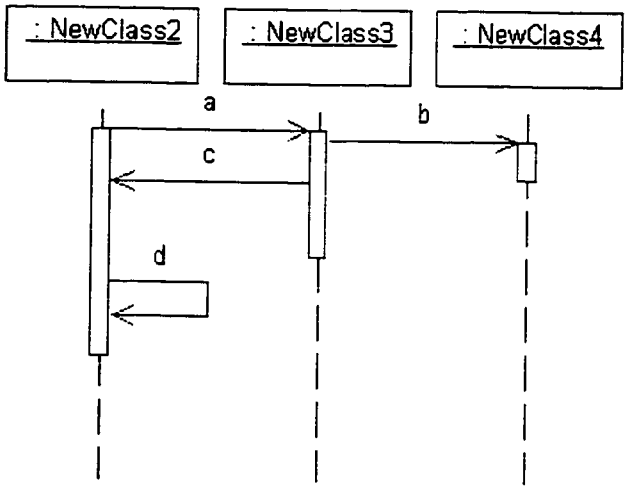

Figure A.3 UML Sequence example diagram

UML collaboration diagrams are similar to the sequence diagrams, but they show the information, in a slightly different way. They use numbers to indicate the temporal relationship of the messages, instead of a timeline. Figure A.4 shows an example of a UML collaboration diagram.

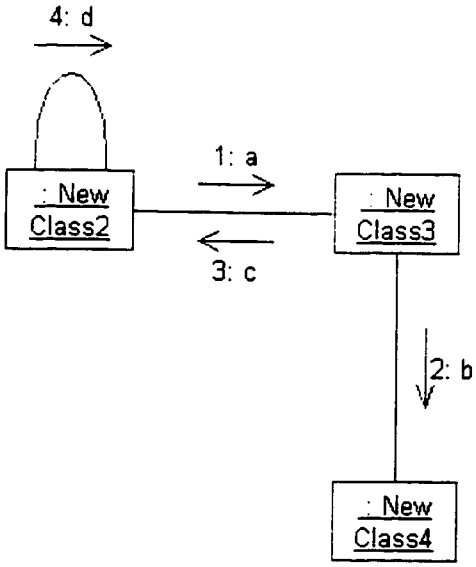

Figure A.4 UML collaboration example diagram

Appendix B

B. Node Data Types and Node Coding Tables

This appendix provides additional information, regarding the way MPEG-4 BIFS encodes the nodes of the scene and their fields. All the information can be found at MPEG-4 Systems specification document [29]. This appendix presents only some examples of the Node Coding Tables (NCTs) and Node Data Types (NDTs).

Table 1 shows the NDT of the IndexedFace Set node. The first column shows the name of the field, while the second the field type. The third on displays the binary number that is used to identify a field at the first decoding of the node (DEF). The next column shows the id that is used to identify a field, which is either an event in or an exposedField field (IN). The fifth column identifies event out and exposedField fields (OUT) while the sixth the fields that can be animated using the BIFS-anim stream.

IndexedFaceSet	SFWorldNode SFGeometryNode					0110000 01001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex	MFInt32		000					
set_coordIndex	MFInt32		001					
set_normalIndex	MFInt32		010					
set_texCoordIndex	MFInt32		011					
color	SFColorNode	0000	100	00				
coord	SFCoordinateNode	0001	101	01				
normal	SFNormalNode	0010	110	10				
texCoord	SFTextureCoordinate Node	0011	111	11				
ccw	SFBool	0100						
colorIndex	MFInt32	0101				[-1, +I]	14	
colorPerVertex	SFBool	0110						
convex	SFBool	0111						
coordIndex	MFInt32	1000				[-1, +I]	14	
creaseAngle	SFFloat	1001				[0, 6.2831853]	6	
normalIndex	MFInt32	1010				[-1, +I]	14	
normalPerVertex	SFBool	1011						
solid	SFBool	1100						
texCoordIndex	MFInt32	1101				[-1, +I]	14	

Table 1 IndexedFaceSet NDT [29]

The information, which is referred at the first line of Table 1 shows the NDT that includes this node. Table 2 is an example of NCT. It shows the SF2DNode NCT. The first column contains the nodes of the NCT and the second their id in binary format. The rest of the columns show the number of bits that is used to identify the fields at the DEF, IN, OUT and DYN formats.

SF2DNode		30 Nodes			
reserved	00000				
Anchor	00001	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AnimationStream	00010	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Background2D	00011	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
ColorInterpolator	00100	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Conditional	00101	0 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
CoordinateInterpolator2D	00110	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
DiscSensor	00111	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
Face	01000	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Form	01001	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Group	01010	0 DEF bits	2 IN bits	0 OUT bits	0 DYN bits
Inline	01011	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
LOD	01100	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Layer2D	01101	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
Layer3D	01110	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Layout	01111	4 DEF bits	4 IN bits	4 OUT bits	2 DYN bits
OrderedGroup	10000	1 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
PlaneSensor2D	10001	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
PositionInterpolator2D	10010	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
ProximitySensor2D	10011	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
QuantizationParameter	10100	6 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
ScalarInterpolator	10101	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Script	10110	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Shape	10111	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
Sound2D	11000	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Switch	11001	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
TimeSensor	11010	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
TouchSensor	11011	0 DEF bits	0 IN bits	3 OUT bits	0 DYN bits
Transform2D	11100	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Valuator	11101	4 DEF bits	5 IN bits	5 OUT bits	0 DYN bits
WorldInfo	11110	1 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

Table 2 SF2DNode NCT [29]

Appendix C

C. Java 3D Hierarchy

This appendix provides some additional information, regarding the architecture of the Java 3D API. Figures are included in [60].

Nodes in Java 3D specifications are divided in two general classes:

- The Group nodes, which include nodes, that can have other nodes as their children.
- The Leaf nodes, which include nodes, that can not have other nodes as their children.

Figure C.1 shows the hierarchy of the Group nodes.

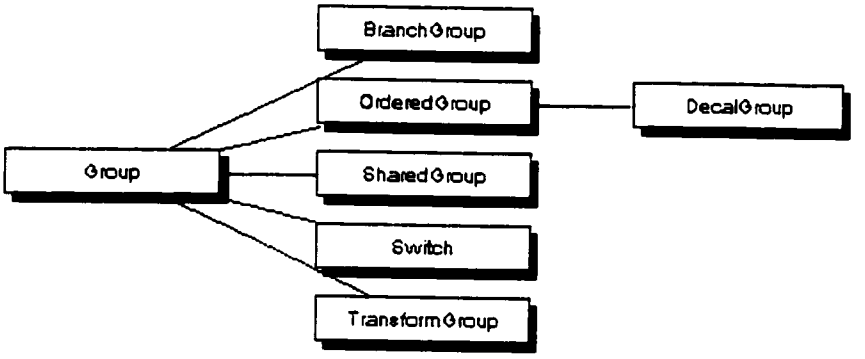

Figure C.1 Group nodes hierarchy

Figure C.2 shows the hierarchy of the Leaf nodes.

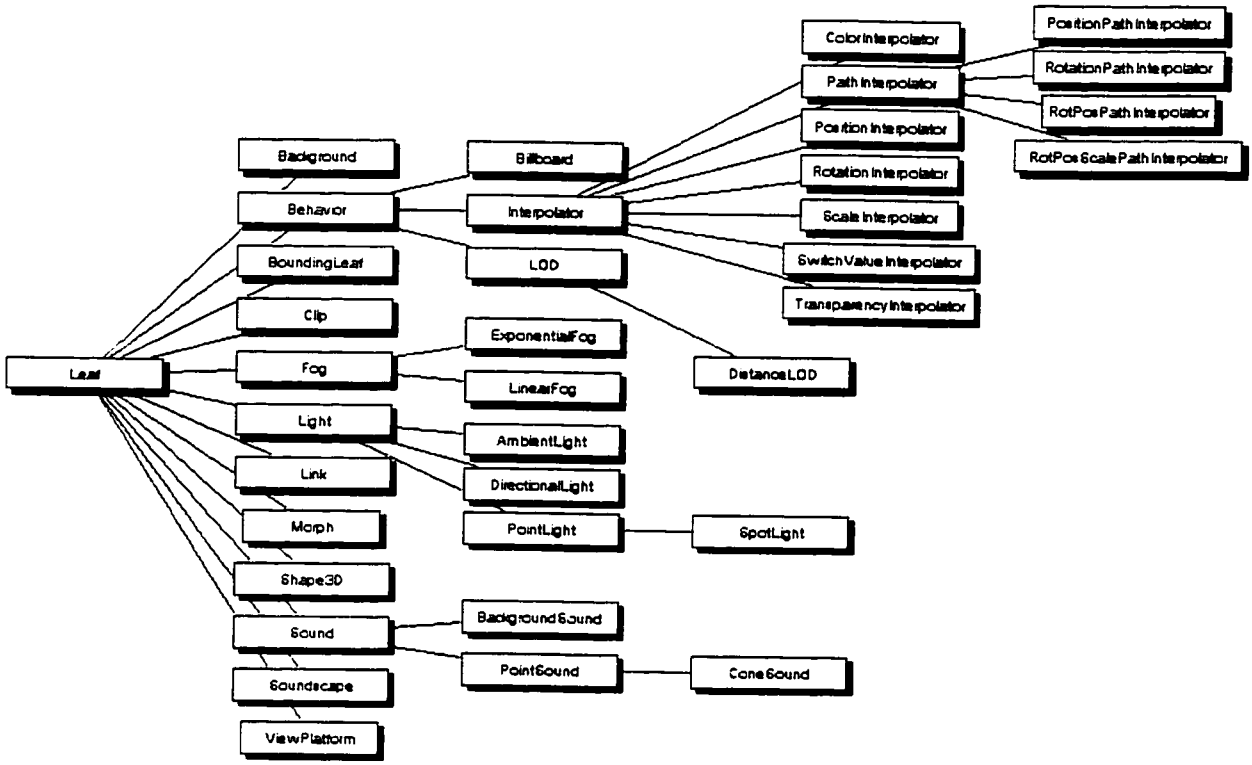

Figure C.2 Leaf nodes hierarchy

Figure C.3 shows the hierarchy of the NodeComponent objects.

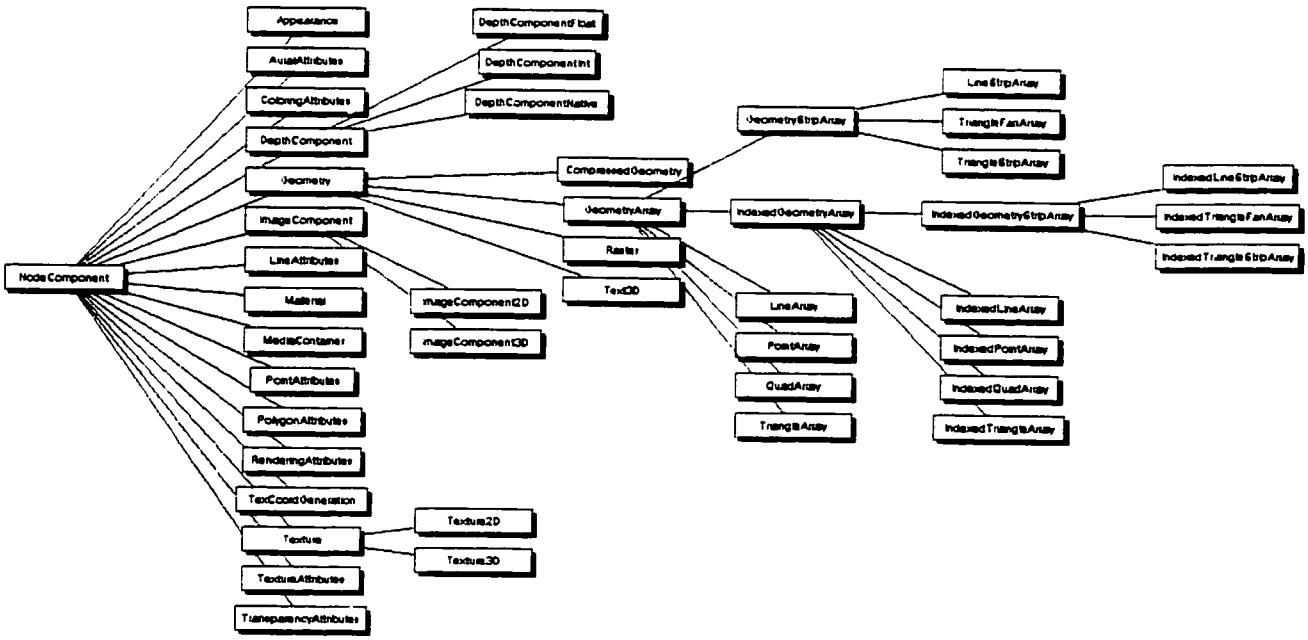

Figure C.3 NodeComponent objects hierarchy

Figure C.4 shows the hierarchy of the Condition objects. They are used to implement MPEG-4 Behavior objects, and to add interaction capabilities at the MPEG-4 Browser (e.g. navigation).

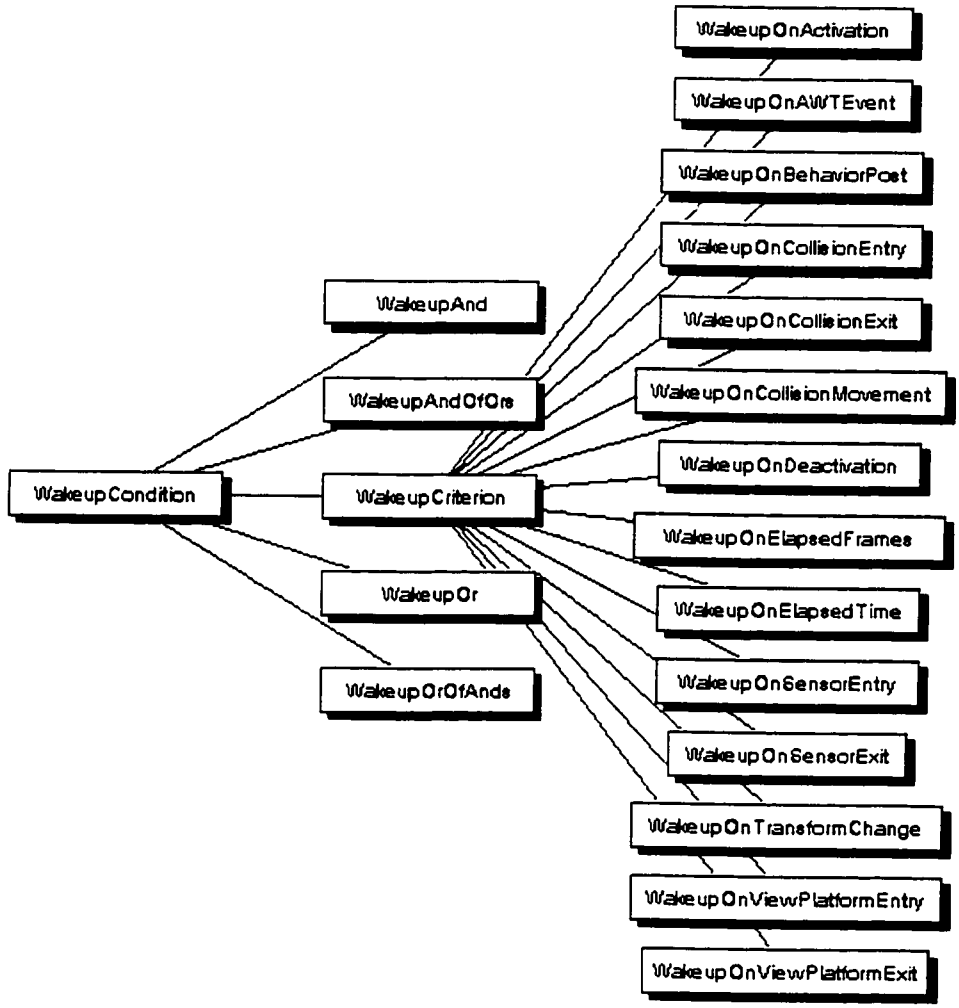

Figure C.4 Condition objects hierarchy