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.

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

Université d'Ottawa · University of Ottawa

An Architecture for Recording and Playback of MPEG4-based Collaborative Virtual Environments

by

Mojtaba Hosseini, B.Sc.

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

A collaborative virtual environment provides the means for different parties to interact with one another and the virtual environment in which they preside and share its space, despite each participant being at a geographically distant location. The development of such an application delves into many areas of research and requires the integration of various technologies some of which are content compression, efficient updating, timely, reliable and efficient transport of content, event synchronization and many more.

MPEG4 is an international standard that deals with some of these issues by putting forward a binary specification of scenes, the updates needed to change the scene, multiplexing natural and synthetic content, synchronization of events and other features that may be used in developing collaborative virtual environments.

This thesis is extending a collaborative framework based on MPEG4 to allow for recording and playback of entire sessions carried out in a collaborative virtual environment and also presents the implementation of an application utilizing such a framework. The work involves the development of a framework supporting MPEG4's Synchronization Layer (SL) to capture events and their temporal characteristics, storing the information according to the standard specification of MPEG4, as well as providing the capability to of playing back the events as they had originally unfolded.

The collaborative framework that will be extended is COSMOS (Collaborative System framework based on MPEG4 Objects and Streams) which provides a basic framework for the development of collaborative virtual environments. Before extending this framework for recording and playback capabilities, there is a need to complement it with additional interaction capabilities so as to have a meaningful recording session. Such additions include new human-computer interfaces, the ability to manipulate arbitrary objects, adding new objects to the scene, and loading and triggering and sharing predefined animation.

Acknowledgement

I would like to thank Dr. Nicolas Georganas for his supervision and continued support and encouragement as well as allowing me to work on this project. I am also indebted to Dr. Shervin Shirmohammady for his moral and mental guidance and the inspiring role model he has provided me with through his dedication and kindness. Francois Malric's contributions to this thesis in the form of many discussions and suggestions were invaluable. Last but not least, my parents and their spiritual support have been instrumental and can never be truly appreciated.

List of Abbreviations

API	Application Programming Interface
BIFS	BInary Format for Scenes
CCD	Client to Client Delay
COSMOS	COllaborative System framework based on MPEG4 Objects and Systems
CTS	Composition Time Stamp
CVE	Collaborative Virtual Environment
DAI	DMIF Application Interface
DTS	Decoding Time Stamp
DMIF	Delivery Multimedia Integration Framework
ESI	Elementary Stream Interface
JMF	Java Media Framework
JNI	Java Native Interface
MPEG	Moving Picture Experts Group
OCR	Object Clock Reference
OTB	Object Time Base
RTP	Real Time Protocol
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
SL	Synchronization Layer
SPS	SL-Packetized Stream
TS	Time Stamp

List of Figures

Table of contents

Chapter 1

1 Introduction

The importance of synthetic media consisting primarily of 3-dimensional object representations, as opposed to natural media such as video and audio, has recently become more pronounced and their communication, compression and processing has attracted considerable focus in the light of their ever increasing applications ranging from military [29], industrial [31], medical simulations [10] to entertainment [6] and information visualization [5]. Although different in some aspects, applications making use of synthetic media must grapple with similar issues as those experienced by the users of traditional or natural media such as video and audio. Efficient and yet easy representation, compression, communication, production, consumption and synchronization of these media are only a few of the areas deserving attention and research. Just as recorders and players of video and audio are major applications of these media, such is also the necessity of recording and playing synthetic media sessions.
This thesis presents an architecture and approach to the recording and playback of a synthetic media session in conjunction with the use of the MPEG4 standard for this purpose. Before proceeding however, a few fundamental concepts of importance need to be introduced.

1.1 Virtual Reality

A term widely used and seldom defined precisely, Virtual Reality (VR) is commonly understood as a synthetic world "created within and sustained by computers and networks, and which users and other objects can inhabit and in which they can interact" [11]. These virtual worlds have traditionally been adapted to military and industrial simulation applications. With the increasing availability of the necessary tools to generate and make use of virtual worlds and the increasing processing power and more

sophisticated and powerful graphical displays, VR has begun to enjoy a more widespread application. While medical training simulations and entertainment were the more natural extensions of the traditional applications of VR, information visualization, virtual conference [27] and collaborative design and engineering [28] are some of the newer adaptations of VR.

1.2 Collaborative Virtual Environments

Just as the sharing of video, audio, still images and other forms of media between geographically distant users has become an important issue, sharing of virtual environments between such users is of great interest. Such a shared space is usually referred to as a Collaborative Virtual Environment (CVE) whose application may be found in collaborative engineering and design, virtual classrooms and conferences, networked games and remote training in various fields such as military, medicine and industry to name a few. CVEs have, in addition to all the VR requirements, the demand that interactions in the virtual world be at least observed by all its participants. This added requirement to share interactions in the form of update messages brings about new issues worthy of consideration and investigation. Reliable transport of update messages, efficient compression of both synthetic objects and their corresponding updates, ensuring consistency between distributed virtual worlds, scalability and synchronizing events are some of such issues. No matter what the application, it is of value to be able to record and thence playback the actions and interactions during a collaborative session in a virtual environment.

1.3 Problem Statement

The ability to record and playback of the actions in a CVE session is of importance in several regards. Just as recording of live audio and video sessions is a way of producing these media, recording collaborative sessions in VR can serve as a way of producing new synthetic worlds [12]. In addition, in the case of CVEs employed for educational purposes, having a record of a virtual lesson is of great use to those not present at a

particular session. They may view the session they failed to attend at a later time and even those who attended the lesson may wish to review it.

A recorded session can also be viewed as a form of latecomer support. A participant joining a session later may have a quick look at the earlier events that were recorded in order to get updated.

The problem addressed in this thesis is adding the ability for interaction in a virtual environment and the recording and playback of the resulting collaborative virtual sessions.

1.4 Proposed Solution

In order to record a collaborative virtual session, there must first exist the capability to share interactions in a shared environment. For this purpose, parts of MPEG4 pertaining to sharing interaction are implemented as part of a framework. The second step is to introduce the architecture to support the specification, encoding and transport of temporal information. This is accomplished by the design and implementation of a framework based on MPEG4's Synchronization Layer (SL).

1.5 Contributions

An outcome of the work presented is a generic framework usable for developing CVEs. An application will also be developed to demonstrate the capabilities of the framework. The work includes the following contributions:

- Design and implementation of an extension to a framework that allows the loading and encoding, according to MPEG4, of animation into virtual scenes and subsequently sharing these animations as well as their triggering in order to show the same animation to all participants.
- Design and implementation of an extension to a framework that enables the application utilizing it to manipulate arbitrary objects in a virtual scene and to make aware other participants of these manipulations by sending the appropriate update messages, in accordance with MPEG4, hence making a shared virtual environment.

- Design and implementation of a framework that will allow the specification and encoding of temporal characteristics in data streams conforming to MPEG4's Synchronization Layer as well as sharing this information in the form of Synchronization Layer packets.
- Design and implementation of an application making use of the aforementioned framework in order to record and subsequently playback a collaborative virtual session.
- A paper, entitled "Suitability of MPEG4's BIFS for Development of Collaborative Virtual Environments", Proc. 10th IEEE International Workshop on Enabling Technologies for Collaborative Enterprises 2001 (WET ICE'01) Cambridge, MA, USA, June 2001.

1.6 Structure of the Thesis

Chapter 2 serves as an introduction to background technologies utilized in this thesis such as the MPEG4 standard and the COSMOS framework. Chapter 3 outlines how the ability to interact in a virtual environment and to share such actions was designed and implemented. Chapter 4 provides the architecture of a system implementing MPEG4's synchronization layer and demonstrates how such a layer is exercised in the recording and playback of a collaborative virtual session, as well as the timely execution of animation. Chapter 5 outlines the prototype implemented to showcase the capabilities of the framework. Chapter 6 reports the results of a performance evaluation test of the system while Chapter 7 and 8 present concluding remarks and discuss issues encountered in this endeavor, respectively.

Chapter 2

2 Background

This chapter provides basic information about the various technologies and concepts closely related to the thesis by giving an overview of each topic. A short discussion of scene description and related issues is presented in order to clarify the work. The MPEG4 standard serves as the core concept behind the thesis and the discussion of relevant sections of the standard will be presented. The Java3D Application Programming Interface is the rendering mechanism, as well as controlling entity of the 3D scenes in use by the framework. A short overview of this programming interface is presented in order to provide the necessary background and terminology for the later discussions of the framework. The implementation of concepts and proposed solutions is done as an extension to a framework called COSMOS whose basic concepts and capabilities will also be outlined.

2.1 MPEG4 Overview

As the title of the thesis suggests, the framework and application designed and implemented make use of the specification of a particular standard. Before introducing the standard, it is worthy to note that constructing a framework founded on an international standard has the advantage of being usable by others as a result of its application-independent characteristic. For instance, if recordings of a session conform to an internationally recognized standard, then they can be viewed and used by entities other than the originator of the recording application as long as both the recorder and the player adhere to the same standard. This is contrasted to an application-dependent solution to the same problem that would exclude the use of the resulting product to only one specific application.

2.1.1 General

The Moving Picture Expert Group (MPEG) is a committee in charge of introducing standards related to multimedia as a part of the International Organization for Standardization (ISO/IEC). Two of the earlier standards introduced by this committee include MPEG1 and MPEG2, which standardized the compression of audio/video data for the use in CD-ROMs and interactive Digital Television [18][19]. One of the more recently finalized standards produced by this group is MPEG4 in which the compression of audio/video and, more consequential with respect to this thesis, synthetic data is specified [20].

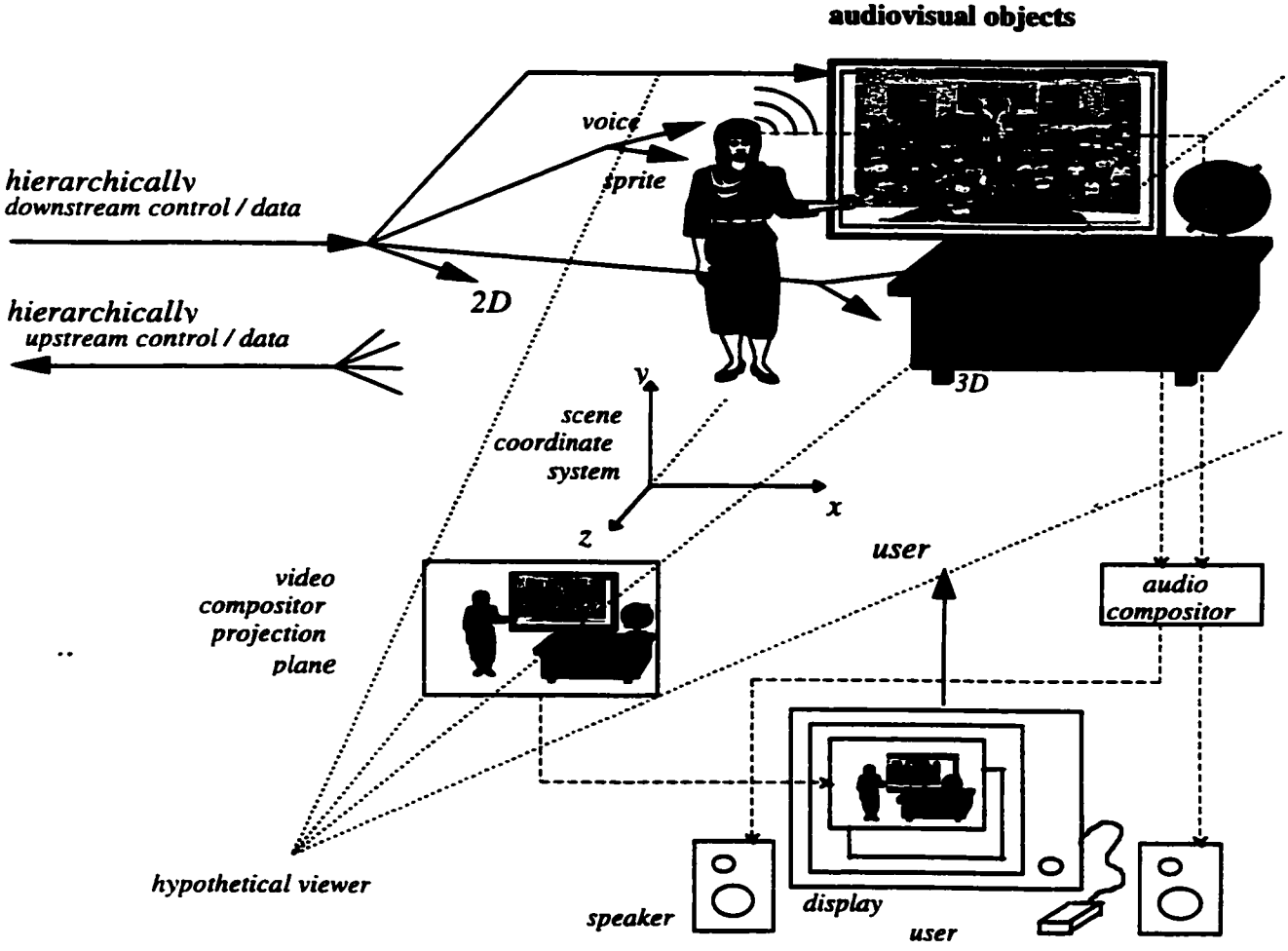


Figure 1. Audio Visual Objects in MPEG4 [21]

In comparison with earlier standards, MPEG4 takes a rather fresh approach to the representation and compression of multimedia by regarding a multimedia product as the fusion of audio/visual objects of possibly different media types in what is called a scene [21]. Figure 1 illustrates this concept.

A multimedia presentation or scene can consist of different forms of media such as audio, video, stationary images and synthetic 3D models, each of which can be regarded as a separate object. Furthermore, each medium may in turn be a collection of objects. In the scene illustrated above for instance, the table and the globe are each a separate object and are both parts of the synthetic media in the scene. Representing scenes on the basis of objects has the following advantages among others:

- Ease of addition and removal of objects in a scene
- Ease of integration of media of different sources and formats into one coherent scene
- Reduction in complexity of accessing individual units or entities within a scene
- Relative ease of providing different versions of the same scene to users with varying capabilities and resources
- The ability to stream updates pertaining to one particular object (e.g. animation for a synthetic object)

The standard then specifies compression algorithms for audio and video media as well as synthetic media based on this object-centered approach.

The specification of the standard is divided into 4 categories, each dealing with different aspect of multimedia compression and representation: Delivery Multimedia Integration Framework (DMIF), Systems, Audio and Video. DMIF provides a delivery application interface irrespective of the underlying network technology [22]. The Systems description of the standard specifies the relationship between audio-visual objects that make up the scene while Audio and Video deal more specifically with the compression of these two natural media. This thesis makes use of the Systems tools specified by the standard, two sections of which are now discussed in more detail.

2.1.2 MPEG4 BIFS

As a part of describing the relationship between audio-visual objects in a scene, MPEG4 standardizes a BInary Format for Scenes (BIFS). BIFS describes the spatial and temporal relationship between objects in a scene and provides a set of 2D and 3D nodes in order to describe its content. This description is closely based on the Virtual Reality Modeling Language (VRML), and shares many of its elements but with the main exception that BIFS is a binary format description and not a textual one, in addition to having more nodes than the standard VRML [23]. Each element in the scene, called a "Node", is described as a part of an acyclic hierarchical graph as shown in Figure 2.

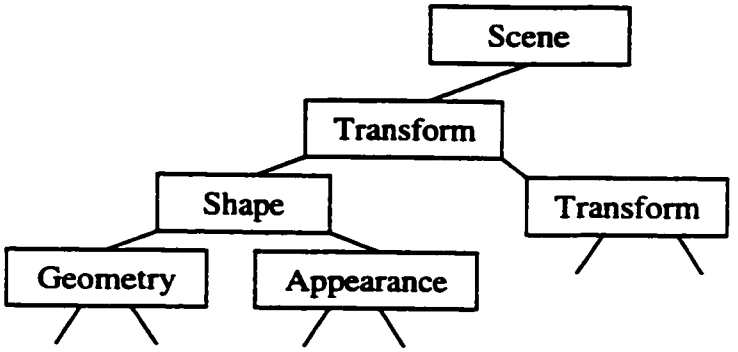


Figure 2. Scene Description Using Nodes

For instance, a "Transform" node is an element that describes the spatial characteristics of an object or groups thereof. Each "Transform" node may in turn have other nodes such as a Shape or another Transform node as its children, which subsequently can have their own child nodes. The hierarchical tree-like structure that results from this description is then encoded into binary, achieving various ratios of compression depending on the scene.

Further compression can be achieved through BIFS' specification of a quantization process applicable to certain fields. In order to make clear the discussion on field quantization in future sections of this, a brief outline will be presented.

Perhaps the more significant feature of BIFS is the ability to access individual nodes within a scene. This ability translates into an easily updateable scene that can subsequently be changed upon receiving a stream of update messages. This method of

updating a scene is accomplished within the context of BIFS-Commands and BIFS-Animation whose discussion will follow that of field quantization in BIFS.

2.1.2.1 BIFS Field Quantization

By default fields in BIFS have a non-quantized encoding but certain types of fields such as 3D positions, 2D positions, color, angle, scale, interpolator keys, rotations and indices can be quantized according to the specification of BIFS. The parameters used in the quantization process are obtained in a special node called "QuantizationParameter". Nodes present within the context of the QuantizationParameter node are quantized according to the parameters specified therein. These parameters include a minimum value, a maximum value and the number of bits used to quantize a field. For instance, for a Coordinate field of an IndexedFaceSet node (see section 2.4.1), all the 3D positions may be contained within the range (100, 5, -40) and (150, 90, 60). In this case, the largest difference is 100 (between –40 and 60) and if a 0.5 accuracy is sufficient, 8-bit quantized values can be used instead of the 32 non-quantized values. By using Quantizationparmater nodes, one can specify parameters best suited for individual objects and hence achieve high compression. The QuantizationParameter node is then treated like other nodes in the scene and has a standard encoding scheme of its field so it can be transmitted to other participants in order to have consistent inverse quantization on all decoding sites.

Once the quantization parameters are known, fields can be quantized either in Intra or Predictive mode. In the Intra mode of quantization, the entire value of the field is quantized while in predictive mode, the difference between the current and last value is quantized. The same quantization process can apply to BIFS access units whose discussion follows.

2.1.2.2 BIFS Command Frames

Referred to as BIFS-Commands, BIFS Command Frames represent access to individual units in a BIFS scene at a given point in time. There are BIFS-Commands for inserting

and deleting elements as well as those pertaining to their replacement. The elements that may be accessed can be nodes or field. While nodes encapsulate units, fields contain information about nodes that they belong to. For instance, a Transform node encapsulates spatial information about an object while a "translation" field of this node gives the position of the Transform node. Using BIFS-Commands, both the Transform node and all its fields including the translation field can be modified. In addition to the 3 BIFS-Command mentioned already (insert, delete and replace) there is a "Replace Scene" BIFS-Command that is used to replace a scene with an entirely new one.

An important feature of BIFS-Commands is that they are also encoded in binary and hence represent small update messages. In order to give an impression of the size of such update messages, it is noted that the non-quantized replacement of a translation field of a Transform node is a 17-byte message. BIFS-Commands can be streamed to provide a continuous update of a node but a better mechanism for such a purpose is the use of BIFS Animation Frames.

2.1.2.3 BIFS Animation Frames

BIFS-Commands are best suited for the conveyance of discrete events since each update is regarded as independent of other updates. There are however many instances where a node or a group of nodes are continuously updated for a period of time. The movements of the lips on a synthetic face or the movements of the fingers of a humanoid in the scene are examples of such updates. When communicating continuous updates, one can take advantage of the inherent correlation between each event to produce a more efficient stream of data. The MPEG4 standard capitalizes on this fact by exploiting the inter-dependency between updates in sequential events or "Animation Frames". Such frames are encoded in the context of BIFS Animation Frames or BIFS-Anims in short. BIFS-Anims provide predictive, as well as intra, modes of quantization for the update values. While employing the intra mode of quantization implies that the absolute value of the update is quantized, the predictive mode indicates that the difference between the current and previous value is quantized instead. Although it includes the word "Animation", BIFS Animation Frames should not be confused with the predefined animation discussed

in section 2.4.2. While the values of regular animations are predefined (specified before they are executed), BIFS-Anims represent a stream of "unpredictable" updates that would normally be generated in real time.

In summary, BIFS-Commands and BIFS-Anims are the two mechanism specified by the MPEG4 for easy and efficient access of elements within a scene for the purpose of their modification. In many instances however, such modifications are required to occur at specific points in time. In order to convey such temporal characteristics, a way of synchronizing the access units must be present which in turn gives way to the Synchronization Layer of MPEG4.

2.1.3 MPEG4 Synchronization Layer

The Synchronization Layer defines the temporal synchronization within and among elementary streams where each elementary stream carries data for the description of a scene or an object within it. The synchronization is accomplished by utilizing time stamps and clock references. Each object in the scene may have its own Object Time Base (OTB). This time base would be conveyed to others by using an Object Clock Reference (OCR). Subsequently, time stamps such as Decoding Time Stamps (DTS) and Composition Time Stamps (CTS) can be used to convey a unique instant in time with respect to a specified OTB for synchronization purposes.

The Synchronization Layer has two interfaces, the Elementary Stream Interface (ESI) and the DMIF Application Interface (DAI) as shown in Figure 3.

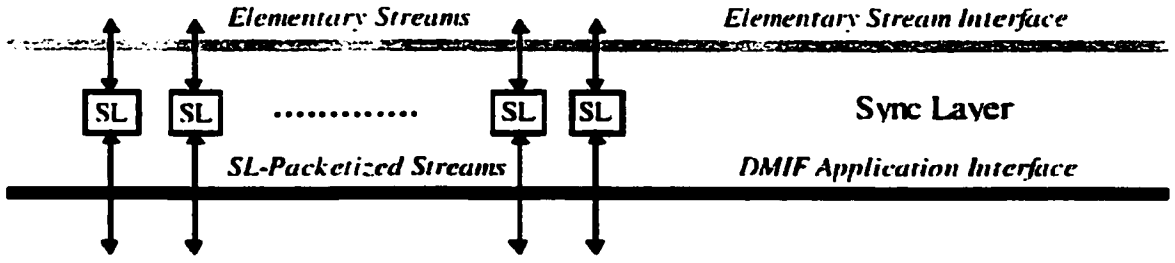


Figure 3. Stream and Delivery Interfaces

The ESI layer specifies the mechanism by which streams are packetized into what are called SL packets. SL packets represent the basic unit for synchronization with the header containing temporal information such as clock references and time stamps and the content containing encoded access units. A sequence of SL packets (called SL-packetized Stream, SPS) is passed on to an underlying layer for delivery through the DAI. Conversely, the Synchronization Layer receives SPSs and must decode the header of each packet and inform the layer above about the various timing information received. It is up to the application to use the timing information provided by ESI.

The specifications of the layer are explained in more detail when discussing the recording and playback of sessions in section 0. However, this layer of the framework is general and application-independent and hence may be used for any application making use of synchronization information.

2.2 Java3D Application Programming Interface

An important, if not the most important, component in a virtual reality application is the display of a 3-dimensional representation of the virtual world visible to the user. The additional capability to control and manipulate the display is also desirable if not necessary. The Java3D Application Programming Interface (API) is one of the options available for this purpose, which provide the building, rendering and control of 3D objects and environments [24]. The Java3D API was chosen in particular for the following reasons:

- As a result of being an extension to the Java programming language, Java3D is constructed on object oriented principles and is also platform independent.
- Java3D provides a high-level programming model, hence removing the requirement for developers to have extensive knowledge of graphics programming.
- Java3D provides the option for the rendering to be done in stereo vision, if the appropriate hardware is available. The ability to display a virtual environment in stereo, which significantly adds to its realism, is an attractive feature of any virtual reality application.

- The availability of built-in features for input devices (such as Head Mounted Devices) and 3D sound in Java3D gives its user the option to scale simple virtual worlds up to more sophisticated environments.
- Java3D being freely available makes it a very economically sound option with respect to perhaps more powerful but expensive software.

The added ability to load VRML geometry and behavior into Java3D via a VRML-Java3D loader, provided by Web3D, is significant since VRML is one of the more popular and available formats for 3D geometry and behaviors [38]. In addition, since the core technology of the framework is Java, interfacing to a Java API is easier and more efficient than using a non-Java rendering tool.

2.3 COSMOS Overview

As mentioned before, the framework proposed in this thesis is an extension of an already existent framework, called COllaborative System framework based on MPEG4 Objects and Streams or COSMOS [8] in short. It is useful to outline the major functionality of this framework in order to clarify the extensions proposed and implemented.

In principal, COSMOS "is an object-oriented framework that augments the rapid development of collaborative applications" and more specifically CVE applications [9]. In short, COSMOS strives to be a common framework used by all CVE application as shown in Figure 4.

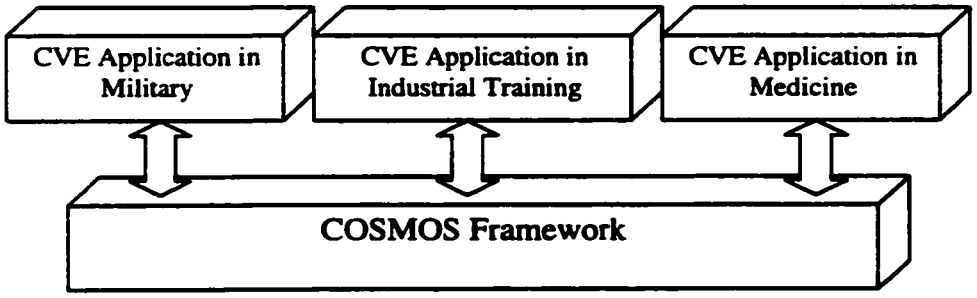


Figure 4. Concept of COSMOS Framework

This framework provides some of the basic and common tools required by CVE applications as sited below:

- Providing the ability to load and render scenes described in VRML or MPEG4 BIFS
- Providing an MPEG4 BIFS encoder/decoder that may be used to compress scenes for storage or transmission
- Providing an interface for the setting up of multicast sessions through the use of MPEG4's DMIF.
- Providing a delivery mechanism for BIFS streams through the use of the Java Media Framework Real Time Protocol Application Programming Interface (JMF RTP API) [25].
- Providing the ability to display a video stream inside a 3D world
- Providing a framework where updates to a scene could be communicated to all participants.

It is important to note that not all VRML and MPEG4 nodes were implemented in COSMOS. For instance nodes pertaining to animation were not supported by the original framework and were added on later, as will be discussed in section 3.3.1. In addition, MPEG4's Synchronization Layer was not a part of the framework and its addition and subsequent use are part of this thesis.

2.4 Scene Description

Any virtual reality application, no matter how simple or complex, must have at its disposal the description of the scene in which the application takes place. In this context, scene description refers to the specification of the temporal, spatial, structural, behavioral and aesthetic characteristics of all the objects in a 3D scene. Generally, a scene is described in terms of a set of predefined nodes, each containing a part of the description. The following is a very short list of common nodes used for scene description:

- Transform: holds spatial information such as position, orientation and scale
- Geometry: holds the geometry of an object most commonly in the form of an IndexedFaceSet.
- IndexedFaceSet: an array of indices and coordinates specifying the geometric shape of an object.
- Material: holds the aesthetic qualities of an object such as color and texture.

The above is only a very small subset of nodes whose combination is used to describe scenes and objects. In most cases, these nodes form a tree-like structure as some nodes take on the role of "parent" with respect to other "child" nodes as already illustrated by Figure 2.

Structuring the scene description in the form of a tree has the benefit of embedding hierarchical information in the description. Hierarchical representation of the scene helps in turn to improve the usability of the description. For instance, moving a Transform node implies the movement of its children nodes as well. As a result, in order to move an object made up of many smaller objects, it is not necessary to apply the movement to every single one but only to the upper most Transform node.

In order to clarify some sections of the thesis, two topics related to scene description are presented in more detail. The description of geometry in an "IndexedFaceSet" node gives an idea of where field quantization can make a significant impact, where as the following section briefly discusses how specific nodes are used to specify temporal and behavioral characteristics of objects.

2.4.1 IndexedFaceSet Node

A node mostly commonly used in VRML to describe a complex geometry is IndexedFaceSet whose description is based on the idea of polygon or triangle meshes [15]. A triangle mesh consists of a group of triangles that together form a surface. These meshes can be used to describe smooth curves provided that the triangles are small enough. The IndexedFaceSet node primarily consists of a list of 3D coordinates ("Coordinate" field) and a list of indices ("coordIndex" field) as shown in Figure 5. IndexedFaceSet Geometry Description .

The Coordinate field specifies the 3D position of points (a1,a2,a3,a4) while the coordIndex field specifies how the triangles are formed using the positions specified in the Coordinate field. For instance, a sequence of (0,1,2) signifies a triangle whose vertices are specified by positions a1, a2 and a3 respectively.

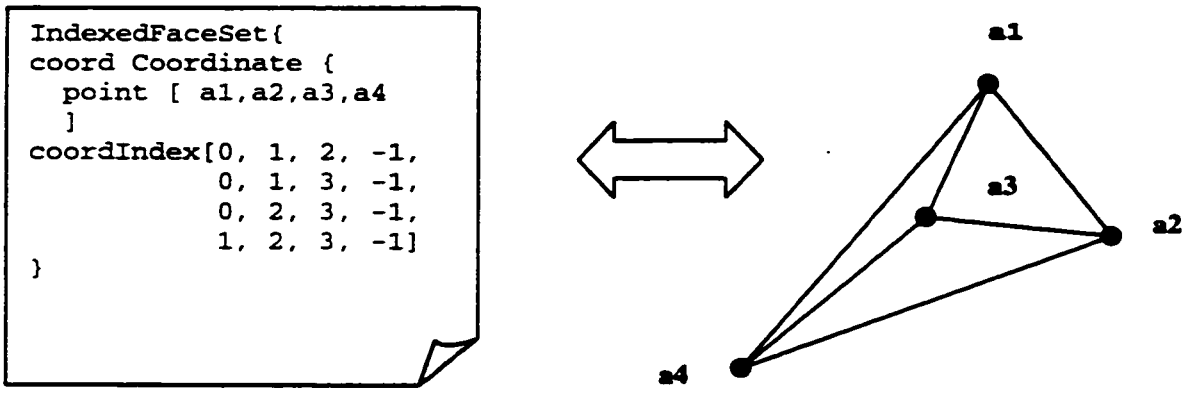


Figure 5. IndexedFaceSet Geometry Description

If no quantization is used, each 3D coordinate in the Coordinate field takes 96 bits (3*32) while each integer in the coordIndex field takes up 32 bits. In the case of complex geometries and especially for curved surfaces, the two fields can include hundreds or even thousands of elements, resulting in large file sizes for descriptions. It is the quantization of these fields that forms the quantization ratios presented in section 6.1.

2.4.2 Animation

Animation is, when used in the context of 3D objects, the predefined change in an object's properties with respect to a temporal spectrum. That is to say, animation is how an object changes (whether in shape, coordinate, aesthetics or otherwise) as time changes. For instance, a 3D humanoid may be animated to walk by defining how every joint of its body moves and rotates with respect to a defined timeline. Animation is used for the definition of known or predefined behaviors of objects. It is predefined in the sense that before its execution, how the object will be changed is known. Consider a fan for instance, whose direction of rotation and its speed are specified or known before it is turned on in a virtual room. The behavior of such a fan can then be animated.

The most common way animation is specified is with the use of "key frames" [1]. In such a scheme, important intermediate steps corresponding to an instant in time in an animation sequence called "key frames" are specified as shown in Figure 6. When executing or displaying the animation, the computer generates the in-between frames based on previous and future key frames.

Figure 6. Animation Using Key Frames

Each key frame corresponds to a value indicating at what point during the timeline of the animation the object should be at a particular position.

Defining animation in this matter requires the specification of the following: a timer (to manage the temporal characteristics of the animation), a trigger (to manage when and how a particular animation is started and stopped) and the key frames.

MPEG4, VRML and Java3D use these elements, albeit slightly differently, to define animation. Figure 7 demonstrates the nodes used by MPEG4 and VRML pertaining to animation.

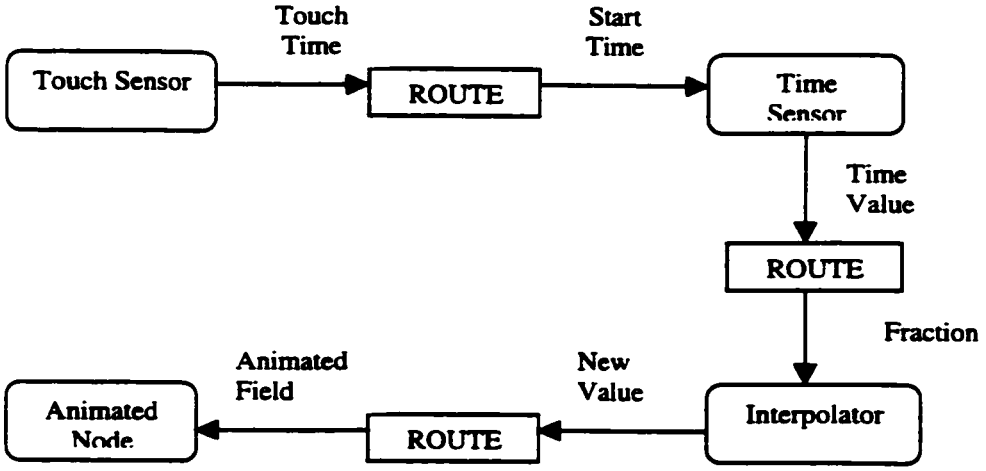


Figure 7. Nodes Defining Animation

The "TouchSensor" node is the trigger point of the animation and is associated with a particular geometry. For instance, a switch may be defined in a virtual room as the trigger point for a fan in a virtual room, hence the TouchSensor node would be associated with the geometry of the node. A "TouchSensor" node detects clicks of the mouse on the object it is associated with and whose "Touch Time" field can be used to trigger the "Start Time" field of the "TimeSensor". The "TimeSensor" node in turn provides a clock for the animation whereby the start time, stop time and speed of execution is specified. The values of the "TimeSensor" regulate the timing of the sequence of key frames generated by the "Interpolator". The "Interpolator" is the node responsible for the timely modification of the properties of an object as well as generating the in-between frames and is temporally driven by the "TimeSensor". The "ROUTE" nodes are the interconnections between the aforementioned nodes.

The Java3D API treats animation in the same essence as VRML and MPEG4 with slight differences in nomenclature and structure. While "ROUTE" nodes connect the various animating nodes in MPEG4, the Java3D equivalent interpolator class must have a handle on the "TimeSensor" (Alpha in Java3D) and the animated node. There is also no "TouchSensor" class in Java3D but a similar behavior class called "PickObjectBehavior" allows the "picking" or selection of objects in the scene. Other subtle differences such as the use of quaternion values in Java3D orientation interpolator as opposed to axis angles in VRML became evident only in the implementation of these nodes.

Chapter 3

3 Providing and Sharing Interaction

As mentioned before, in order for a collaborative session to be meaningful, its users must have a great deal of freedom in the virtual environment. This freedom manifests itself in the users having an impression of existing in the virtual environment and their ability to influence their surroundings, be it manipulating objects or introducing, making, breaking or eliminating them. Such interactions with the virtual world are especially important if the resulting session is to be recorded for playback. As a result, the addition of interaction and animation to the COSMOS framework was deemed a necessary precursor to the recording and playback of the session.

3.1 Interaction in a Virtual Environment

Humans interact with the real world around them in a variety of ways, employing their different senses and faculties. This interaction can be thought of as belonging to two main categories: a person's influence on his/her surroundings and the influence of the surroundings on him/her. The goal of a virtual environment is to ultimately provide its users with the same and perhaps even more ways of interactions as those with the real world. However, much of this aim is yet to be realized and so far most VR applications provide only a basic interaction capability. One of these basic capabilities is the manipulation of position and orientation of objects in a virtual scene and the other, navigating in such an environment.

The COSMOS framework provided the navigation capability but the ability to manipulate arbitrary objects was added as a part of the work done for this thesis in addition to enhancing the navigation. Both of these interactions are briefly discussed in order to illuminate the kind of collaborative session that was used for the recording.

3.1.1 Navigation and Presence

In order for the virtual environment to be somewhat realistic, the users must have a presence in the scene and also be able to move around and explore or navigate in it. In a collaborative virtual environment, all other participants must observe the presence of every other user within their sight. For this purpose, a particular geometry called an "avatar" is usually associated with each user as their embodiment in the shared virtual space [34]. The COSMOS framework allows for arbitrary objects to be chosen by the users as their avatars. A database of already existent geometries is available to choose from but the framework is generic and abstracts the geometry of the avatar from the functionality. There are constraints on the shapes that can be used to represent a user however. For instance, if the user's hand movements are to be tracked, the avatar representing the user must have a "hand" geometry, hence a simple "stickman" shape will no longer be sufficient.

As a part of this thesis, improvements were made to the presence aspect of the framework since that is one area where COSMOS was in need of improvement. The enhancements with regards to presence were twofold: conveying the movements of the head of each avatar and tracking of the hand and fingers of the users and mapping them to those of the avatar.

As mentioned previously, one of the ultimate goals of VR is to represent in the virtual world the users as they are in the real world. This includes movements of the user's body parts, be it the head, hands, shoulders or on a more minuscule level, face muscles whose combination of movements convey an expression [14]. It was towards this goal that the movements of the hand of the user were tracked and represented in the virtual scene and subsequently shared with other users. In addition, the movements of the user's view were mapped to the head of the avatar and shared with others.

3.1.1.1 Avatar Head Movements

The movements of the head of a user might at first seem unimportant in a collaborative endeavor. Although this might be the case relative to the movements of other body parts such as the hand, the movements of the head of a user can be used to convey important

information and hence improve collaboration. For instance, in a virtual conference scenario, head movements of participants may be an indication of where their attention lies. When there is more than one person talking, this becomes all the more significant as the direction of the head of a listener can be used to infer whom he/she is paying attention to. During an education or training session, the movements of the head of the trainer or educator performing a task, can be an important parameter in deciding where the attention and focus lie in the procedure. For these reasons, it was decided to map the movements of a user's viewpoint to that of the head of the avatar representing that particular user as shown in Figure 8.

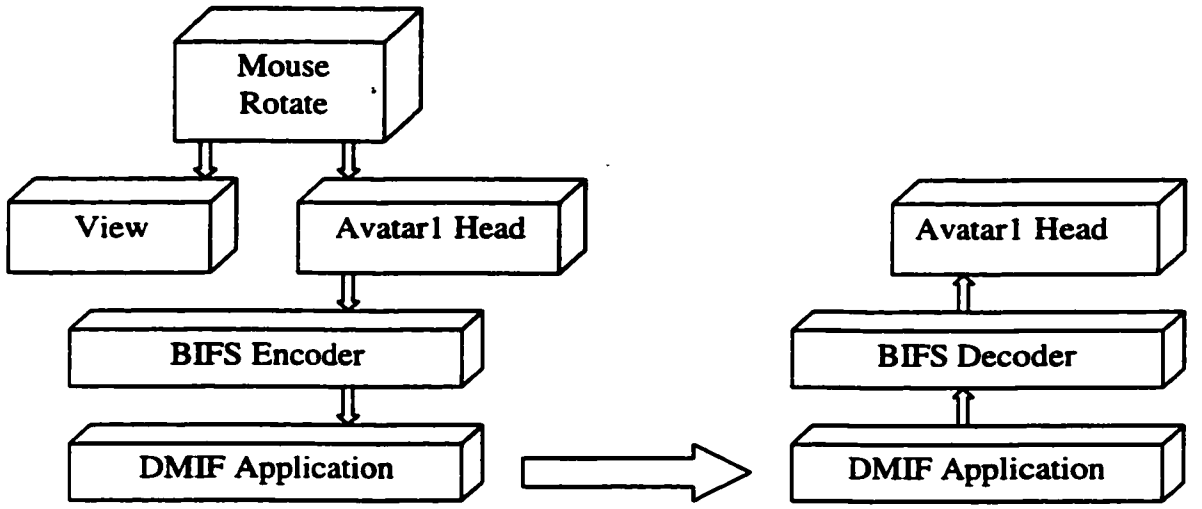


Figure 8. Sharing Avatar Head Movements

It should be mentioned that recent studies have shown that the movement of the head of avatars in a collaborative task is not enough to convey focus of attention even in conjunction with pointing gestures of the arm, but sharing this information can be seen as a first step towards improving collaboration in virtual environments [16].

A few points about the implementation are worthy of mention. The "Mouse Rotate" component is a Java3D built-in class that provides general rotation capabilities. These capabilities were used to rotate both the view of the user and the geometry representing the head of the user so as to give the impression that the avatar's head moves with the movements of the user's view. The details of delivery of the head movements of an

avatar are outlined in section 3.2. At this point, it suffices to state that every participant is informed of the new head rotation through MPEG4 BIFS update messages.

3.1.1.2 Tracking User's Hand and Finger Movements

An important parameter in developing VR applications and/or frameworks is the human-computer interface or in other words how a user interacts with the virtual world. Simple VR applications usually make use of traditional input devices such as the keyboard, mouse or joystick. Users of such applications may however find them restrictive and limited. This is especially the case if the application involves manipulation of objects in the virtual scene. A human user manipulating the real world with his/her 10 fingers can not possibly feel just as comfortable interacting with the virtual world with a 3-buttoned mouse. The research involving new forms of human input devices is ongoing and a few commercial input devices have already emerged for this purpose. Two of these devices were used as a part of this thesis to track the movements of the user's hand and fingers and map them to the corresponding avatar geometry. The addition of these input devices into the framework not only enhances the virtual experience but also provides a more meaningful session for recording as a result of more events occurring.

3.1.1.3 Tracking Finger Movements

In order to manipulate the virtual environment with one's fingers, the movements of those fingers in the real world must first be measured and then mapped to the finger geometry of the avatar in the virtual scene. For the purpose of tracking, the Virtual Technologies Inc.'s [36] CyberGlove™ device was used. A C++ interface is made available with the device, however, since the COSMOS framework is in Java, it was necessary to write a generic Java interface for this device using the Java Native Interface (JNI) as shown in Figure 9. JNI is the mechanism by which Java programs can communicate with and use code written in C++ [26].

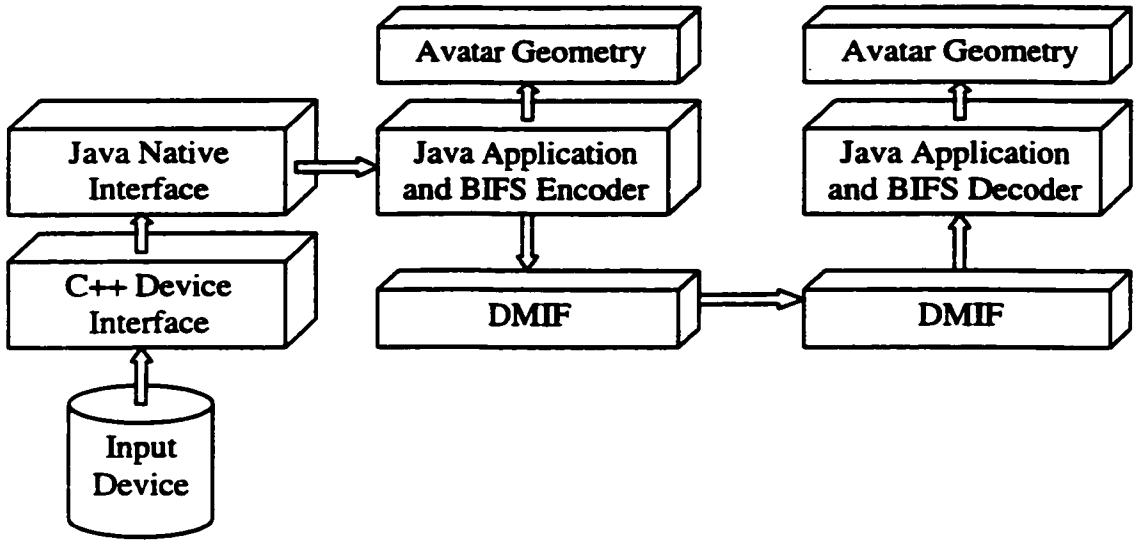


Figure 9. Using Input Devices

JNI serves as the interface between the Java application and the C++ device interface. The values received from the input device are then mapped to the geometry representing the local user's hand and fingers. At the same time, such information is conveyed to all participants in order for them to change the sending avatar's hand and finger positions so as to share these movements. The delivery of all updates is discussed in more detail in section 3.2.

The C++ device interface of the CyberGlove™ produces 20 rotation values, each pertaining to a different joint on the human finger as shown in Figure 10.

Element	*0*	*1*	*2*	*4*	*5*	*6*	*9*	*10*	*11*	*13*	*14*	*15*
Joint	Base	mid	top	base	mid	top	base	mid	top	base	mid	top
Finger	Thumb	thumb	thumb	index	index	index	middle	middle	middle	ring	ring	ring

17	*18*	*19*
base	Mid	top
pinky	Pinky	pinky

Element	*3*	*7*	*8*	*12*	*16*
Abduction	Not used	Not used	Index and thumb	Middle and ring	Ring and pinky

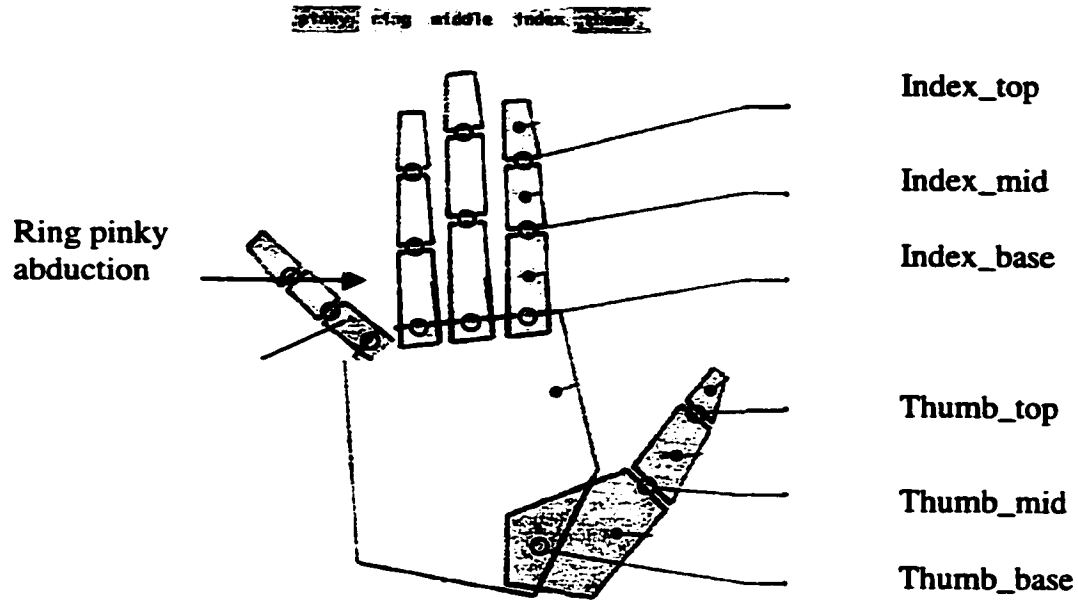


Figure 10. Nodes in Hand Geometry

A VRML hand geometry [3] built according to the H-Anim [17] standard was used as the avatar's hand and upon receipt of these 20 rotation values, the VRML model was modified to reflect the movements of the user's fingers. These values were then transmitted to all other participants in the form of MPEG4 BIFS-Anim update messages.

3.1.1.4 Tracking Hand Movements

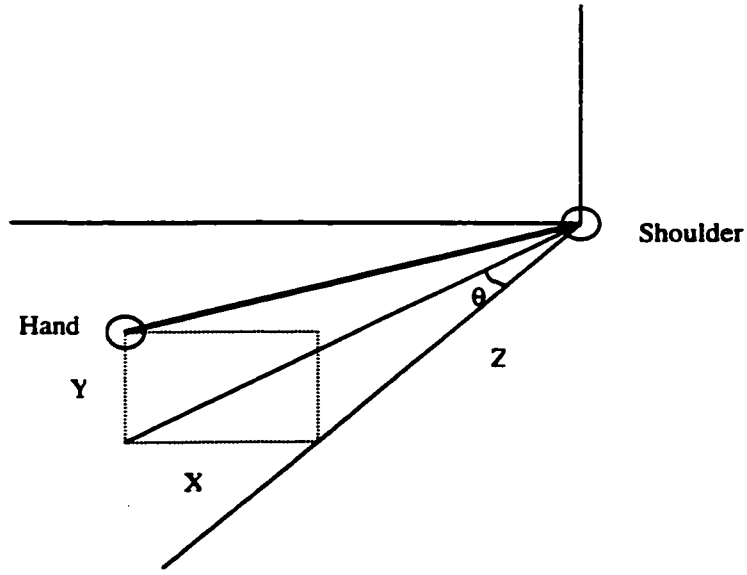
In order to track the movements of the user's hand, a general-purpose tracker miniBird™ device was used. Similar to the finger-tracking device, JNI was utilized in order to write the Java interface for the miniBird™ device [2]. The interface provides the application with 6 values reflecting the 6 degrees of freedom (3 for translation and 3 for rotation). These values are then mapped to the VRML geometry representing the user's hand as well as being transmitted to other participants of the session.

3.1.1.5 Producing Shoulder and Elbow Movements by Inverse Kinematics

As can be imagined, a hand moving in 3D space without being attached to a body can produce an eerie effect on the viewers and that was precisely the result of tracking the hand of the user and not other body parts. While a solution is to track the shoulder and elbow corresponding to the hand, hence producing the need for more trackers, a simpler alternative is to use inverse kinematics to calculate the elbow and shoulder rotations that would result in the current position of the hand.

The problem of inducing the rotation of an elbow and shoulder joint based on the position of the hand is well known in the field of robotics. While complete and general solutions to this problem have been well-documented in various robotics books [30], a simpler solution was deemed more appropriate for this particular case to reduce the computation. The simpler solution is based on the assumption that the movements of the hand will be restricted to the front of the user. That is to say that the user is assumed to never bring his/her hand behind the body. The assumption was made to simplify the calculations since the aim is not a complete inverse kinematics model but a way of connecting the hand to the body for a simple application. Following this assumption, one can derive the rotation components of the elbow and shoulder joint as follows.

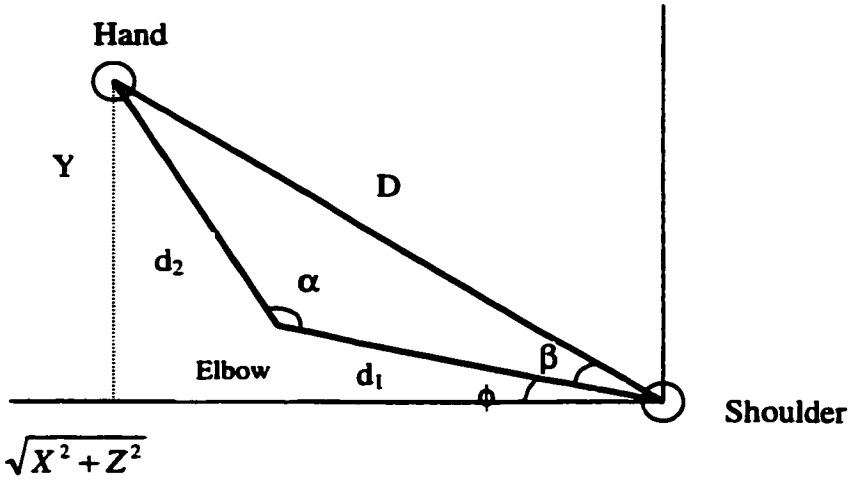
The rotation of the shoulder in the y-direction is first calculated as seen below.



θ = Rotation of the shoulder joint in the y-direction

$\theta = \tan^{-1}(X/Z)$

The rotation of the elbow joint is then calculated.



D = Distance from shoulder to hand

d_1 = Length of arm (shoulder to elbow)

d_2 = Length of forearm (elbow to hand)

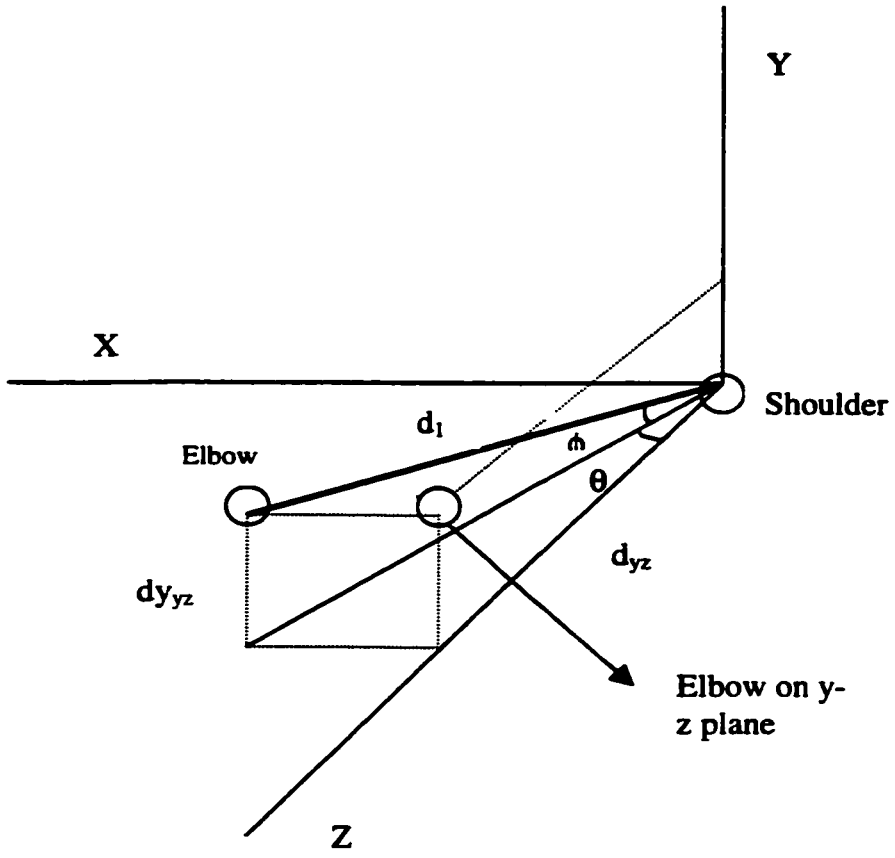
α = Rotation of the elbow

$$D = \sqrt{X^2 + Y^2 + Z^2}$$

$$\alpha = \cos^{-1}\left[\frac{(D^2 - d_1^2 - d_2^2)}{(-2d_1 d_2)}\right] \qquad \text{(Cosine Law)}$$

$$\beta = \sin^{-1}\left(\frac{d_2}{D}\sin\alpha\right) \qquad \text{(Sine Law)}$$

$$\phi = \tan^{-1}\left(\frac{Y}{\sqrt{X^2 + Z^2}}\right) - \beta$$

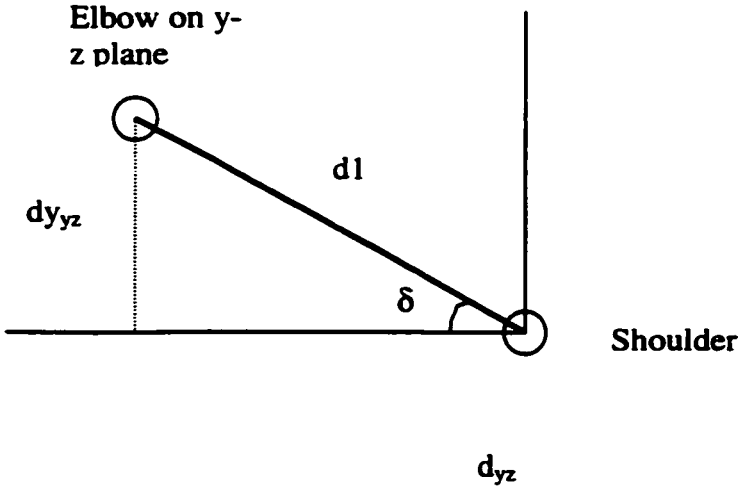


The last part is to calculate the rotation of the shoulder joint with respect to the z-axis.

$dy_{yz} = d_1 \sin\phi$

$d_{yz} = d_1 \cos\phi \cos\theta$

The projection of the elbow on the y-z plane is first calculated as shown above. The shoulder rotation with respect to the z-axis is then calculated as shown below.



δ = Shoulder rotation in the z-direction.

$$\delta = \tan^{-1}\left(\frac{dy_{yz}}{d_{yz}}\right) = \tan^{-1}\left(\frac{\tan\phi}{\cos\theta}\right)$$

The three angles calculated (θ,α,δ) are then used to move the shoulder and elbow of the avatar so as to follow the position of the hand.

The result of the hand and finger tracking as well as the inverse kinematics calculations can be seen in chapter 5.

3.1.2 Manipulation

As discussed before, in addition to the presence of users in the virtual world, their ability to manipulate their surrounding environment forms the second category of interactions. The manipulation of the virtual environment can include actions such as changing the geometry of objects in the scene (e.g. breaking objects), modifying the aesthetics of the scene (e.g. coloring objects), introducing or eliminating objects in the scene or moving objects in the scene (e.g. picking up an object). While the COSMOS framework enabled the introduction and elimination of objects in the scene and sharing these changes, the ability to move arbitrary objects in the scene was added as a part of this thesis. This is simply done by making use of Java3D's picking and moving behavior classes (Figure 11).

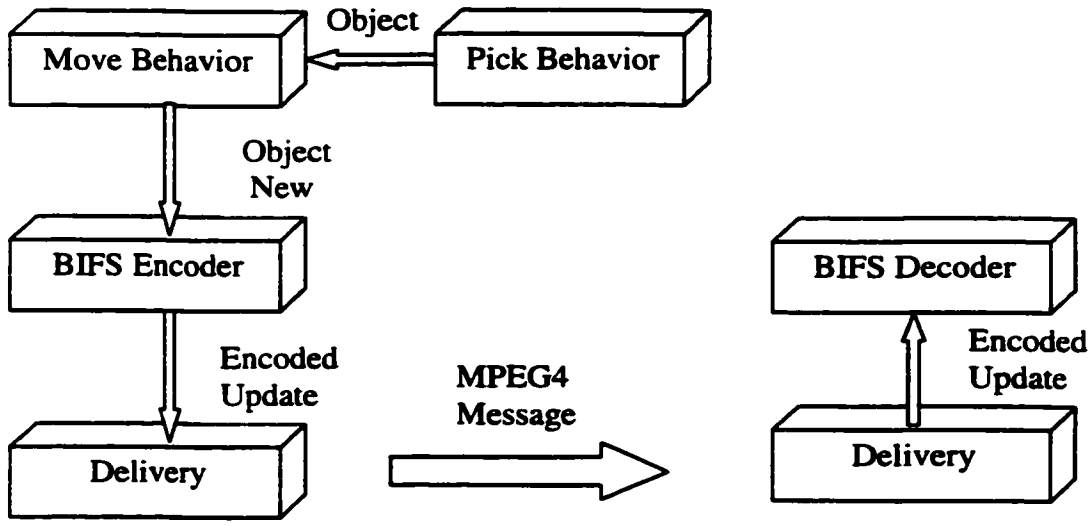


Figure 11. Object Manipulation

Java3D's pick behavior identifies the object that the user wishes to manipulate, while the move behaviors accomplish the manipulation. The application is informed of identity of the object being manipulated and the new position or orientation values. This information is then encoded in terms of either a BIFS Command Frame or BIFS Animation Frame and passed on to the delivery layer to be sent to other participants.

A few points are worthy of mention at this point. The capability outlined above allows remote users to manipulate objects in a shared virtual environment. However there are no "locking" mechanisms to ensure that while one user is manipulating an object, another is denied access to it, hence providing mutual exclusion to the shared object.

The framework abstracts how objects are manipulated from the sharing of the manipulation. For instance, if the hand tracking mechanism presented in section 3.1.1.2 is extended to allow the manipulation of objects through the use of the cyber glove, then the pick and move behaviors are replaced but the encoding interface remains the same. This abstraction allows the framework to grow in future versions with regards to the human-computer interface without necessitating changes in lower layers.

3.2 Sharing of Interaction

The essence of a CVE lies in sharing all the interactions of its users with all other interested participants so as to give an impression that they all preside in the same place. As mentioned in section 2.1.2, the MPEG4 standard defines two ways of updating scenes: BIFS Command Frames and BIFS Animation Frames (BIFS Anims). The previous version of the COSMOS framework had implemented the BIFS Command Frames and used it to update the virtual scene. While BIFS-Commands are necessary to introduce new objects to the scene or eliminate them, most other types of interactions added so far are best encoded in BIFS-Anims. Before discussing the reasons for this, it is best to distinguish between discrete and continuous events in the virtual world. Discrete events are those that happen infrequently where as continuous events happen on regular or repeated intervals. Examples of discrete events in the virtual world are the introduction of a new object to the scene, replacing a scene with an entirely new one or the triggering of a predefined animation. Examples of continuous events are the movements of an avatar's

various body parts such as lips, hands, fingers or facial muscles. Keeping these concepts in mind, one can discuss the possible uses of BIFS-Commands and BIFS-Anims.

3.2.1 BIFS-Command versus BIFS-Anims

While BIFS-Commands and BIFS-Anims both provide access to or updates of a scene description, their differences make them better suited for different types of updates. For every BIFS-Command update, the command type, identification of the node being updated and the specific updated field along with new value of the field are encoded for transmission (Figure 12).

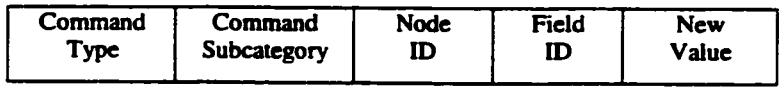


Figure 12. BIFS Command Frame

Figure 13 shows the command types and their subcategories in a BIFS-Command update. While 2 bits are needed to specify both of these parameters the number of bits used for the remainder of the update message is implementation dependent. As will be discussed in section 3.4, every node that may be updated is given an identification tag (ID) that is uniquely associated with that node. The number of bits used for the node IDs is implementation dependent. In addition, the number of bits used to encode the new value of a field is also left by the MPEG4 standard to the implementation. For instance in an implementation of the standard using 32 bits for node IDs and the default non-quantized 32 bits to represent float values will result in a BIFS-Command of 17 bytes for an update to a translation of an object. Quantization has a significant impact since the quantized version of the same BIFS-Command occupies only 8 bytes.

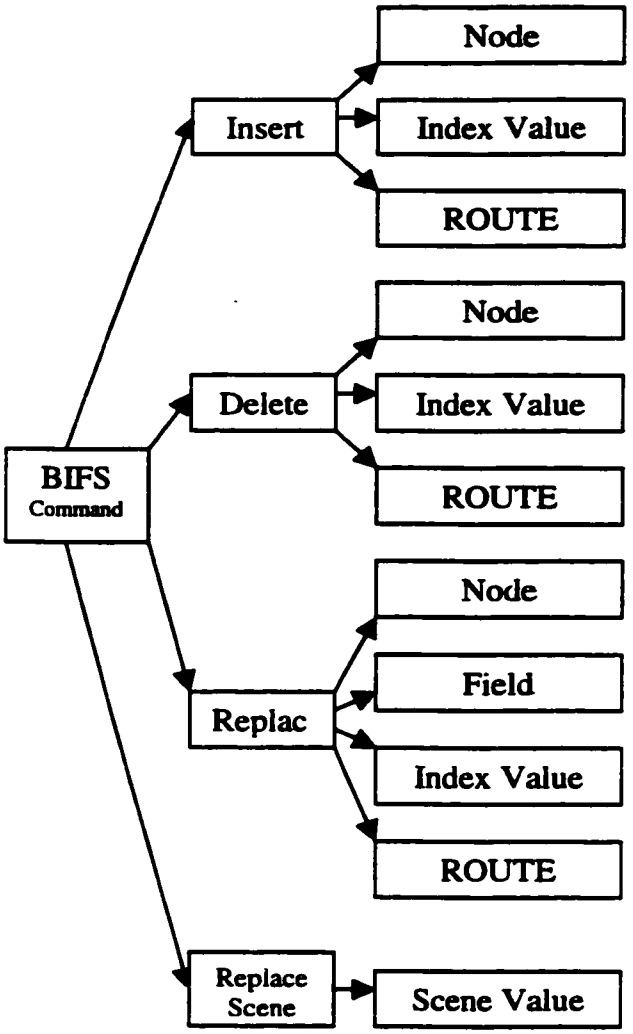


Figure 13. Types of BIFS-Commands

Conversely, update messages in the form of BIFS-Anims are encoded in a different manner. The encoding of BIFS-Anim updates is done in two stages: configuration followed by the updates (Figure 14).

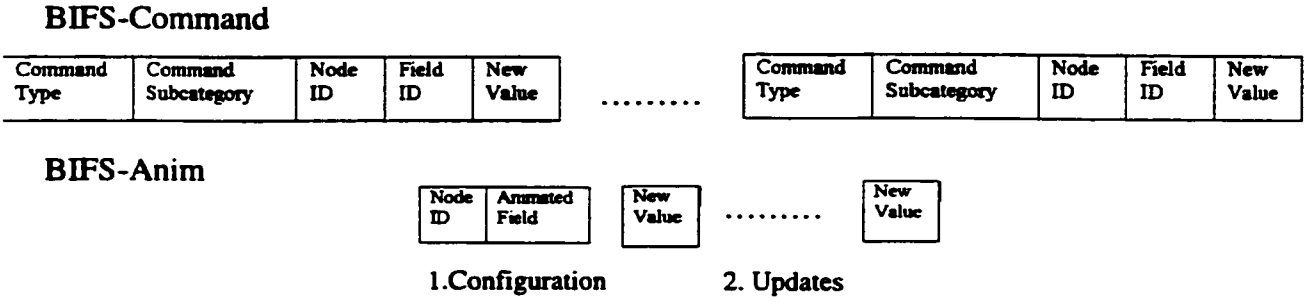


Figure 14. BIFS-Command and BIFS-Anim Comparison

The configuration of a BIFS-Anim stream occurs once and carries stream-redundant information to all participants. This information includes a list of IDs belonging to nodes that will at some point be animated and the corresponding animated field or fields. The actual updates in the form of Animation Frames then only carry the new value and a few bits specifying which of the animated nodes are currently animated (Figure 15).

Configuration

Header	NodeID1	which field	NodeID2	Which field	...	NodeIDn	Which field

Update

Header	Is Active	New value	...	New value

Figure 15. BIFS-Anim Configuration and Update

It should be mentioned that other information such as quantization parameters may be sent in the configuration but the diagram above illustrates the case when default quantization parameters are used. In such a case, the configuration parameters include a list of node IDs pertaining to a list of nodes that will be animated and the identification of their animated field (e.g. node ID of a Transform Group and its translation field). This information is sent only once but may be updated at later stages in the session. The updates then only include a small header and an identification of which of the animated nodes are currently animated and the new value. For instance, if the configuration contained a list of 4 nodes to be animated, only 4 bits are sent to indicate which of the 4 nodes are to be animated (the "is Active" field) in the current message (e.g. A sequence of 0100 would indicate that the second node is animated).

Examining the two types of updates, one can observe that BIFS-Anims are more efficient with respect to BIFS-Commands when applied to continuously animated nodes. This is because in BIFS-Anims the node IDs and the animated fields are sent only once as opposed to sending them with every update message, which is the case in BIFS-Commands. This reduction may not seem significant at first. However it should be kept in mind that in a realistic and advanced virtual reality applications, a great number of nodes would be animated simultaneously. Tracking the joints on a human body (and not

the facial muscles) can easily result in more than 40 continuously animated nodes. Coupling this with the fact that many individuals may be present in a virtual scene brings into light the significance of smaller updates in the case of BIFS-Anims.

3.2.2 Implementation of BIFS-Anims

The general guideline for the encoding of BIFS Animation Frames was presented in the previous section. The Java implementation of a framework supporting the encoding involves the hierarchy of classes shown in Figure 16.

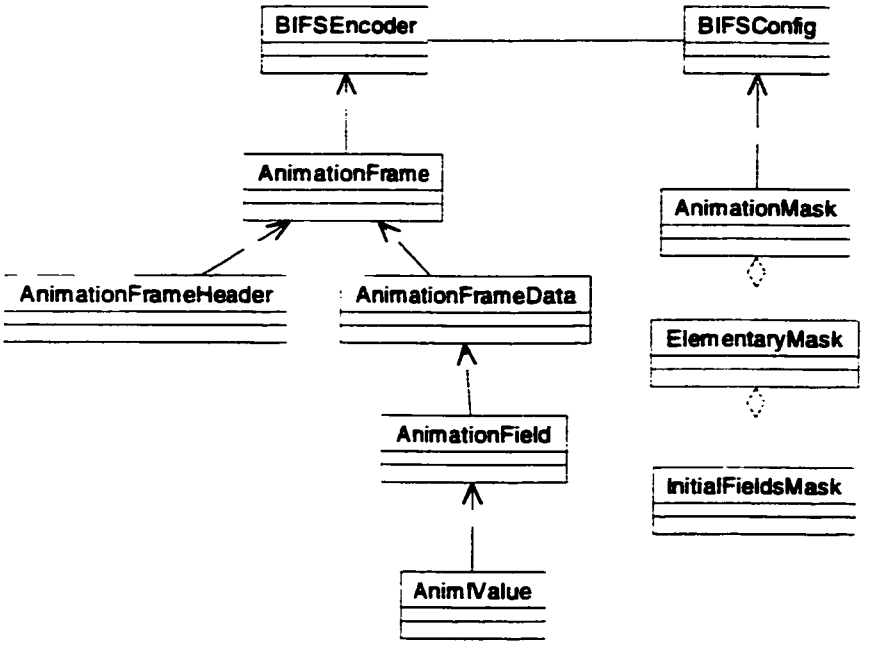


Figure 16. BIFS-Anim Implementation Class Hierarchy

Two separate groups of classes can be observed from above, each fulfilling a separate function. "BIFSConfig" holds all configuration information relevant to the BIFS Encoder and Decoder. The part of this information relating to the encoding of BIFS-Anim streams is contained in the "AnimationMask" class which holds a list of "ElementaryMask" classes, each containing configuration information about one animated node. Each "ElementaryMask" in turn contains a list of "InitialFieldsMask" classes pertaining to the

encoding configuration of individual fields. As mentioned before, the configuration information is sent once as shown in Figure 17.

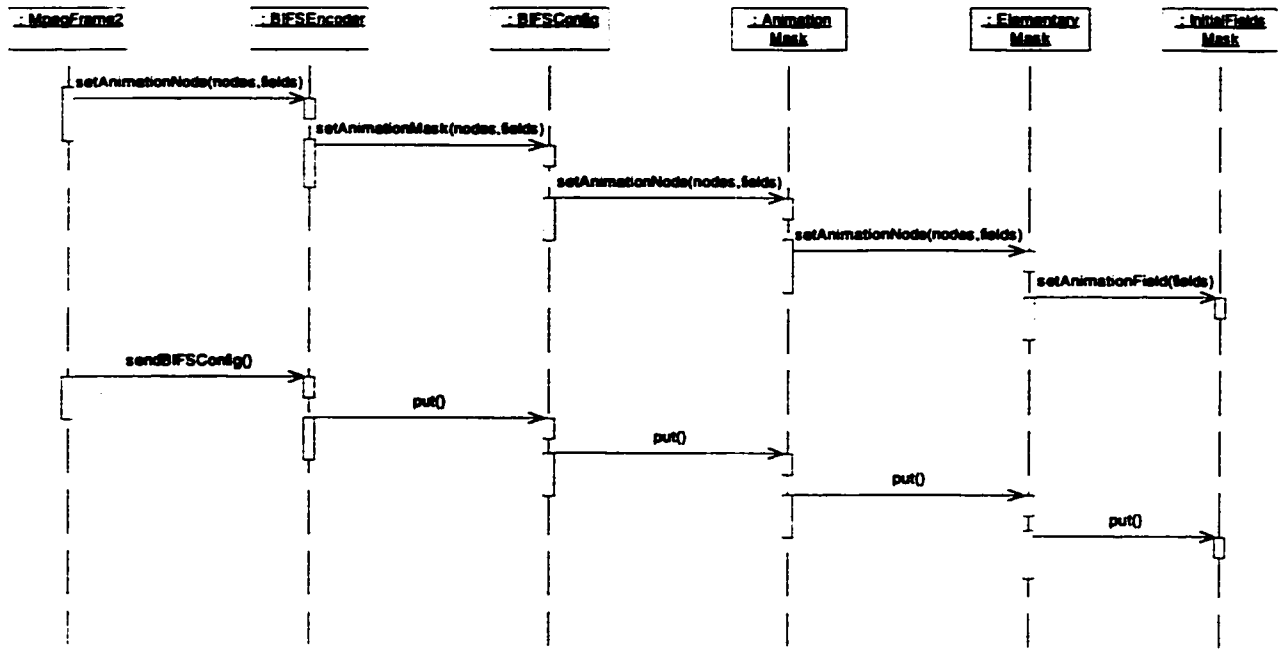


Figure 17. Encoding BIFS-Anim Configuration Sequence Diagram

"MpegFrame2" represents the application making use of the framework. It passes the list of nodes to be animated along with their perspective fields to the BIFS Encoder who configures the "BIFSConfig" class. When the application wishes to multicast the BIFS configuration to all participants, the BIFS Encoder encodes this information and passes them on to the delivery layer.

Once all terminals have the same BIFS Configuration, the application can proceed to animate any of the designated animation nodes (those previously passed to the BIFS Configuration) as illustrated in Figure 18.

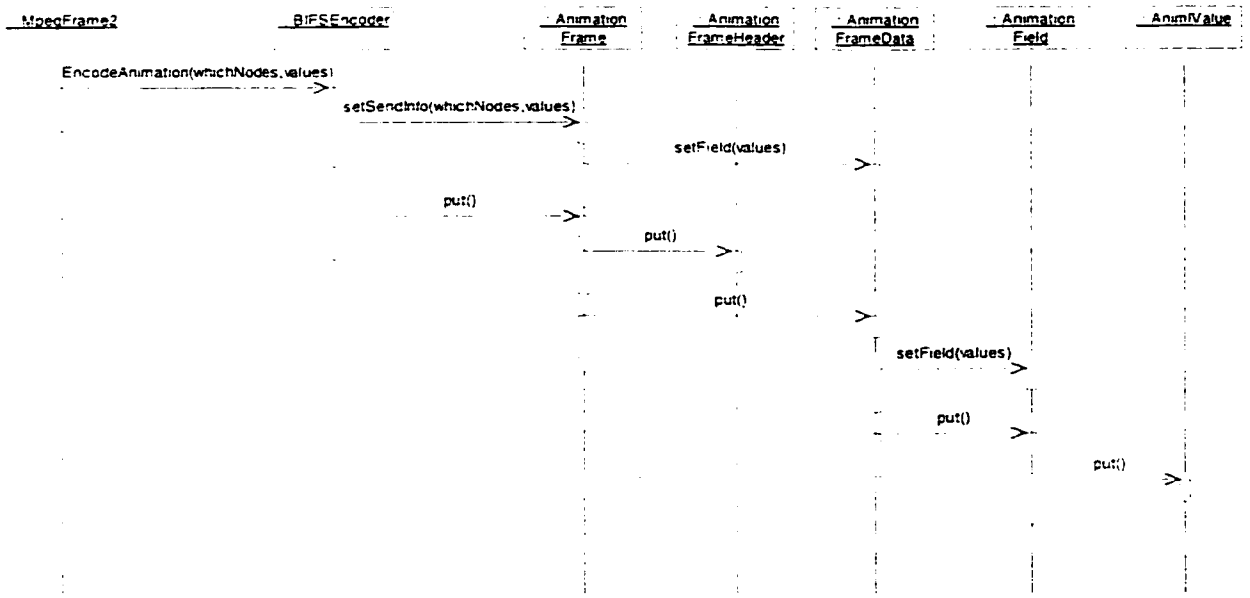


Figure 18. Encoding Animation Frame Sequence Diagram

The application passes the identity of currently animated node(s) and their new values to the BIFS Encoder which in turn sets the appropriately parameters and proceeds to encoding the BIFS-Anim.

3.2.2.1 Quantization

A very attractive feature of MPEG4's BIFS specification is the quantization algorithms that it puts forward. Quantization can be of significant use in the description of 3D scenes and their subsequent updates and manipulations. Scene descriptions usually contain large arrays of floating point numbers as a part of describing detailed objects and meshes. The position and orientations are also represented by arrays of floating point numbers. As a result, the quantization of all these numbers can result in further compression of the scene description. In addition, BIFS update messages usually contain a stream of new position and orientation values and hence can also be reduced in size by quantization.

The quantization of BIFS-Anims was implemented, achieving a reduction in the size of each update. For instance, if it is assumed that the minimum and maximum values in the

update of position vectors is (-10,-10,-10) and (10,10,10) respectively, one can use 8 bits to represent each element of the vector instead of the regular 32 bit representation of floating point numbers hence achieving a 4:1 reduction in size. The minimum and maximum values sited above are reasonable, assuming that objects and avatars in the virtual world are confined to rooms of 20 by 20 units. However, if this assumption ceases to hold, one can easily update the quantization parameters of the encoder and decoders via an update to their configuration.

3.3 Sharing Predefined Animation

The concept of predefined animation was introduced in section 2.4.2. It is customary for virtual environments to make use of animations in order to add realism and livelihood to the scene. The explosion of a house, the passing of a car, the changing of street lights and rotation of a fan are only a few examples of where the behavior of objects within the scene can be specified as animation. As a natural extension of virtual environments, a CVE also includes animations and must therefore be able to share them.

3.3.1 Loading Animation

Before sharing the animation, the framework must have the definition of the animation. BIFS specifies all the necessary nodes (as introduced in section 2.4.2) for the definition of animation and their binary encoding. In order to implement these nodes as an extension of the COSMOS framework, the gap between the Java3D and VRML specification discussed in section 2.4.2 had to be reconciled. For instance, Java's Observer interface was used as a replacement of the "ROUTE" nodes in VRML, an idea already implemented in the previous COSMOS framework. It is therefore of use to present the concept of Java's Observer interface briefly.

The Observer interface in Java allows objects to "observe" other objects, hence allowing one object to be notified of another object's change. For instance, if object A is an observer of object B, object A's Update method is called when object B's Notify method

is invoked. The observer interface can be used to replace the "ROUTE" nodes since it essentially fulfils the same function.

With the use of the Observer interface, loading of animation is performed in 2 steps:

1- Node Implementation: Each of the animation related nodes are implemented in Java3D. The parser constructs the equivalent classes in Java3D of the MPEG4 nodes: Java3D's "PickObjectBehavior" as the "TouchSensor", Alpha as the "TimeSensor" and Interpolator as the "Interpolator". It should be mentioned that the "PickObjectBehavior" is attached to its parent group node, hence making all the objects below that node the trigger points of the animation.

2- Node Connection: Either of the Observable interface or variable passing replace "ROUTE" connections in MPEG4. The "TimeSensor" node's start time field becomes an observer of "TouchSensor" node's touch time field and the field of the animated node becomes an observer of Interpolator's "value changed field". The Java3D implementation of the "TimeSensor" also becomes an observer of the "TimeSensor" node's start time field. In addition, Interpolator is passed the Alpha implementation of the "TimeSensor" and the animated node (usually a TransformGroup).

Once the scene and the animation have been loaded, the animation may be triggered by clicking on the trigger points discussed in step 1 above, resulting in the following chain of events:

1. "PickObjectBehavior" is woken up and updates the "TouchSensor" node.
2. "TouchSensor" node's touch time field is set to the current time, hence notifying "TimeSensor" node's start time field.
3. "TimeSensor" node's start time field is set equivalent to the "TouchSensor" node's touch time field, hence notifying the Java3D implementation of the "TimeSensor".
4. The Alpha class is started, triggering the interpolator in Java3D and therefore the animation as a whole. The application at this time has the chance to transmit the change in the update if it is a part of an ongoing session. The details of the transmission are discussed in section 3.3.2.

The above steps can be illustrated in Figure 19.

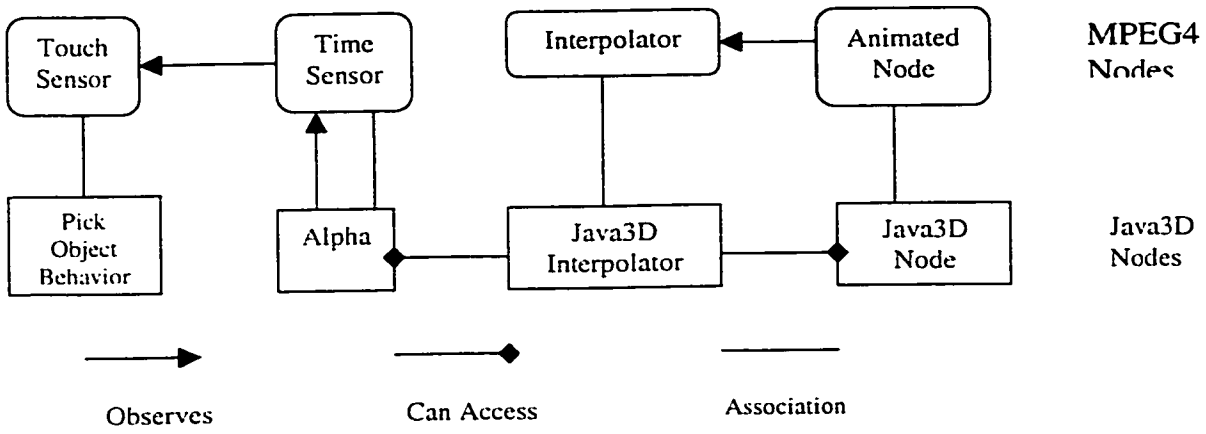


Figure 19. Java3D Animated Related Nodes

As noted in the diagram, the Alpha class is started once it receives an update from the "TimeSensor" node. Since the Java3D interpolators are directly associated with their corresponding Alpha object, the starting of the Alpha results in the interpolator taking effect and commencing the animation of the appropriate node as shown in Figure 20.

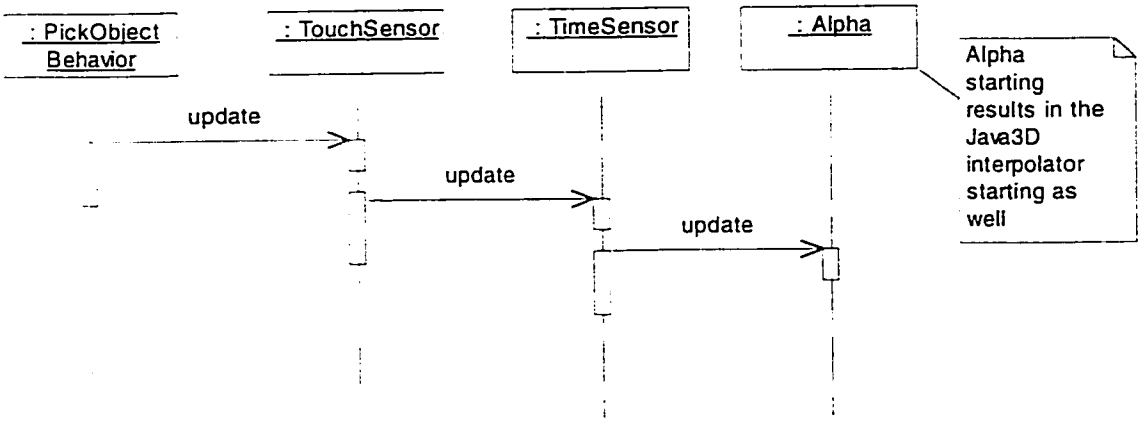


Figure 20. Triggering Animation

3.3.2 Sharing Animation

The animation nodes are encoded in the same manner as other BIFS nodes and sent with the scene or object, to which they belong, resulting in all the participants having all the animation information locally. The only operation that remains in order to share this animation is the sharing of its triggering. Once one of the participants starts the animation

on one station, an update message in the form of BIFS-Commands is sent to all the participants informing them of the change in the start time of that particular animation. The animation begins to execute once the update message is received and decoded. There is however no locking mechanisms at this point to ensure that only one user can start/stop animations.

This scheme of sharing the trigger works as long as there is no time difference between the system clocks on various stations, a highly unlikely assumption. As a result, a mechanism must exist to synchronize the time basis of the animated object on all machines in order for the animation to be triggered at an appropriate time. This mechanism will make use of a synchronization layer whose implementation is discussed in section 4.2.

3.4 Scene Management

As specified by the MPEG4 standard, nodes are to be accessed by a number representing their unique identification called their "node ID". Unlike VRML, a number is used as the identification as opposed to a string name but similar to VRML there needs to be a decision as to which nodes should be given an ID and hence be made accessible. Making all nodes in the scene accessible greatly adds to its complexity and size. Conversely, having no accessible nodes removes the possibility of interaction and change of the scene. Therefore, a reasonable balance needs to be reached depending on the application.

An important part of a framework allowing virtual collaboration is the management of the scene and especially the accessible nodes within it. For this purpose, a Scene Manager was introduced in the original COSMOS framework whose functionality was extended in order to accommodate additions to the scene.

As shown in Figure 21, whether a scene is loaded from the local file or received through the delivery layer and the BIFS decoder, the Scene Manager is informed of the list of accessible nodes and their IDs. Subsequently, whether an object is added to an existing scene or received from the delivery layer through the decoder, its accessible nodes are also passed on to the Scene Manager. As a result, the Scene Manager acts as a central component for all the CVE application's inquiries about accessible nodes. For instance,

when the application modifies an object in the scene, it queries the Scene Manager about the ID of the node corresponding to that object. The application can make use of the ID by encoding an update message regarding that particular node via the BIFS encoder.

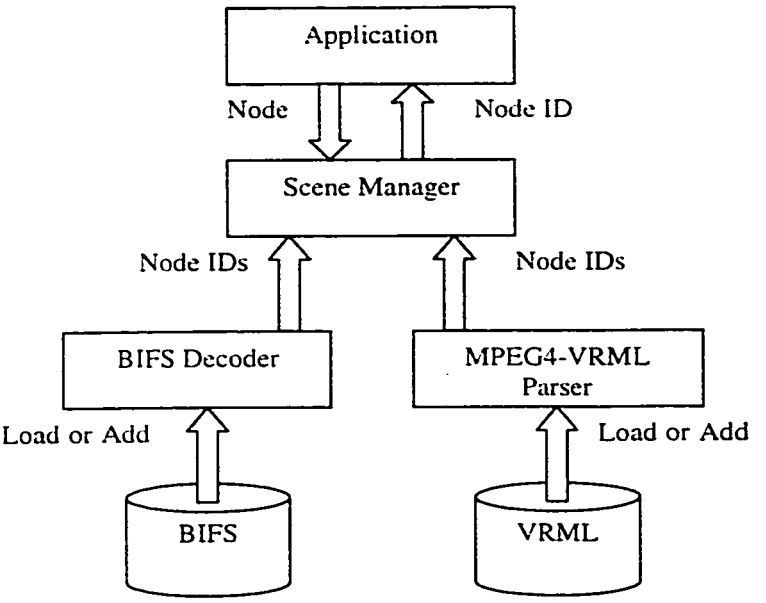


Figure 21. Scene Manager

Chapter 4

4 Recording and Playback

The recording of video and audio in its different forms has become common place. It might however be difficult at first to conceive how synthetic media can be captured. With regards to video and audio, recording implies the registering of visual and acoustic events in the real world by a device. With respect to synthetic media however, the events occur not in the real world but in the virtual world. As the virtual world more closely mimics that of the real world, the recording of its events gains importance. Some of the applications of synthetic media recording have already been mentioned in section 1.3. Having developed an interactive virtual environment, the applications of recording its events serve as a motivation to develop a framework that considers timing information and allows for the recording and subsequent playback of synthetic media.

4.1 Conceptual Architecture of the Framework

The architecture of the framework developed strives to adhere to the MPEG4 standard's Systems specifications. The overall conceptual architecture is shown in Figure 22.

The application is generic and may be any type of collaborative virtual environment or, as we shall see later, a recorder/player of collaborative sessions. The application makes use of the framework by passing scene information along with timing parameters. Scene information is a general term and encapsulates the description of the scene and all subsequent updates to the scene. Timing parameters define the synchronicity of scene information relative to one another or to a predefined clock. 2.1.3 introduced the main timing parameters such as Object Time Base (OTB), Object Clock Reference (OCR), Decoding Time Stamp (DTS) and Composition Time Stamp (CTS). These and other synchronization related parameters are passed to the framework along with the data that

they describe. Conversely, the application receives scene information and the corresponding timing information from the framework.

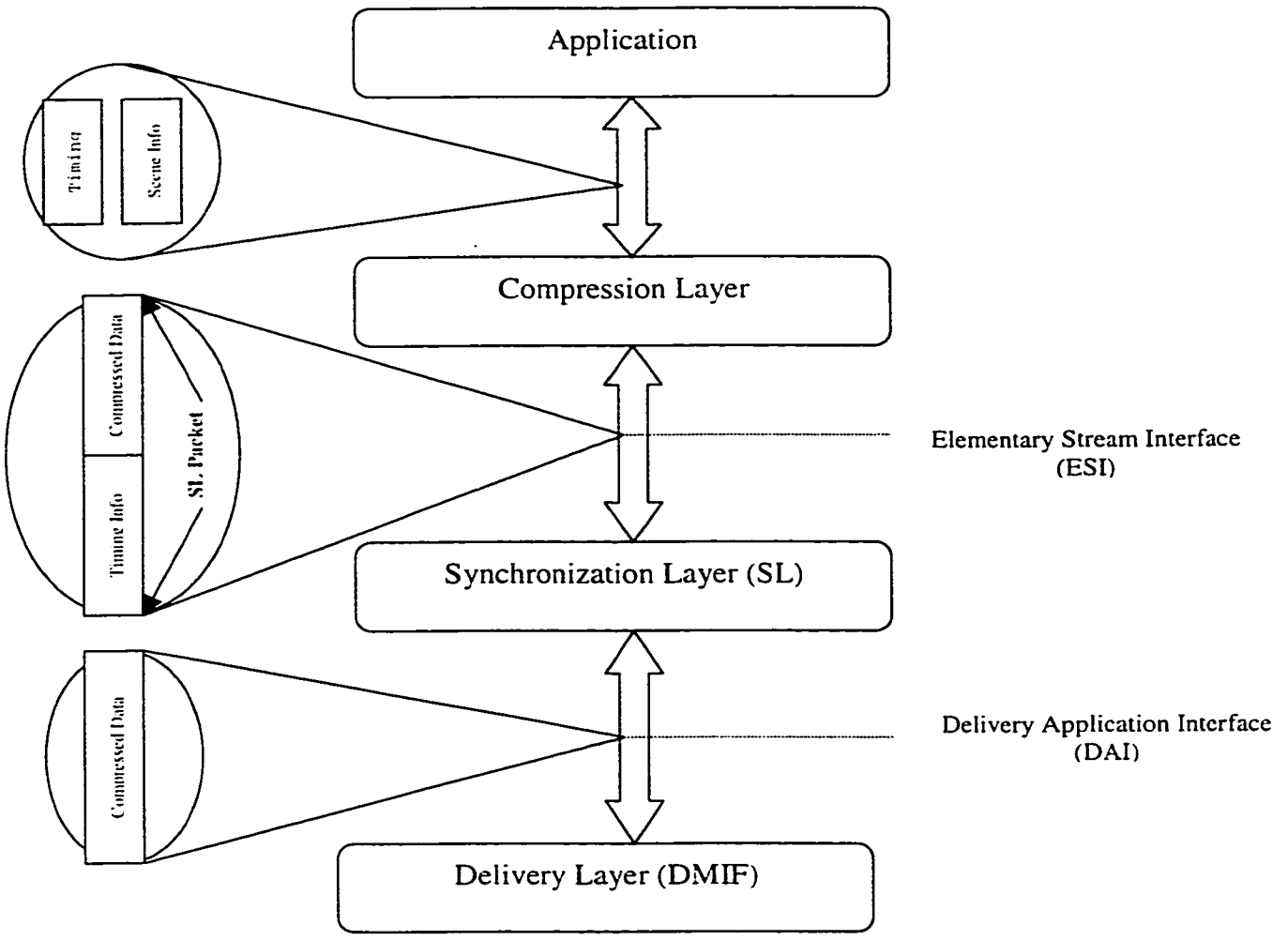


Figure 22. Overall Architecture

The first layer of the framework, called the Compression Layer, encodes and decodes scenes into BIFS. In addition to scene information and any other object information, the Compression Layer allows the compression of update messages into BIFS-Commands and BIFS-Anims. The improvements to this layer were discussed in section 3.2.2 as well as section 3.3. The Compression Layer exchanges compressed data with its lower layer

via the Elementary Stream Interface (ESI) without being aware of timing parameters. It simply passes timing parameters as received from the user to the Synchronization Layer.
The Synchronization Layer is used to convey temporal synchronization within and among elementary streams where each elementary stream can represent a stream of update messages. This layer encodes and decodes the timing parameters already discussed in accordance with the MPEG4 standard. In essence, this layer converts SL-Packetized Streams to elementary streams and vice versa. The details of this process are discussed in section 4.1.1.
The Synchronization Layer communicates with the delivery layer via the DMIF Application Interface (DAI) and hence is delivery unaware. The delivery layer wraps the SL packets in the lower layer protocol (RTP in this case) and multicasts it to all other participants. Conversely, upon receiving data from the network, the Delivery Layer retrieves the SL packets and presents them to the Synchronization Layer. It should also be mentioned that retrieval from and writing to the local directory are also done through the Delivery Layer.

4.1.1 SL Packets

SL Packets encapsulate the elementary stream but also contain encoded timing information. Similar to the BIFS Frames, the conveyance of timing information according to the MPEG4 standard has a main configuration component, called the SL Configuration Descriptor. The Configuration Descriptor determines how SL Packets are to be encoded and decoded by specifying the number of bits used to send clock references and time stamps, the resolution of such references as well as the number of bits used to specify the numbering of packets (for ordering purposes). The default values for these parameters can be present locally but changes can be transmitted during the session. Figure 23 illustrates the various SL configuration parameters.

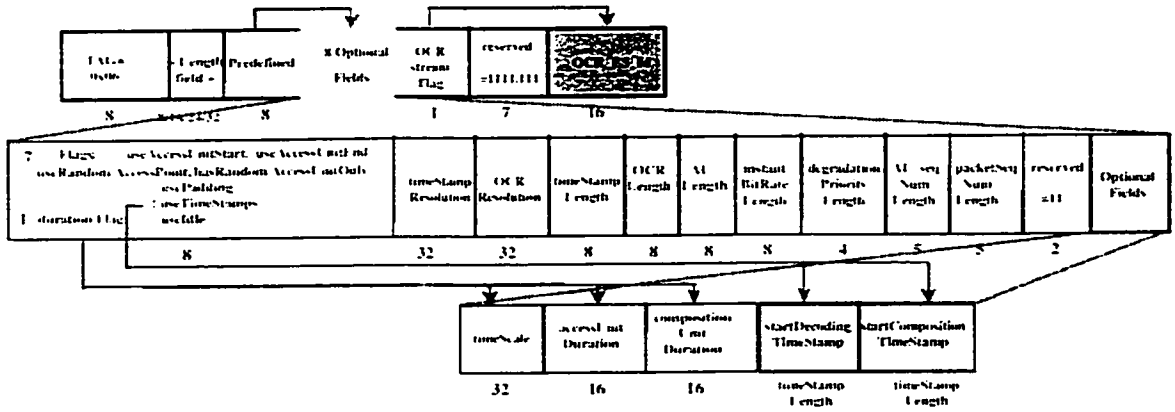


Figure 23. SL Packet Configuration

Not all of the parameters shown are of interest but some of the more important ones include OCR Resolution, Time Stamp Resolution, OCR Length and Time Stamp Length. OCR and Time Stamp Resolutions specify the accuracy of the Object Clock Reference and Time Stamps respectively in units of ticks per second. For instance, an OCR Resolution of 1000 indicates accuracy in milliseconds. OCR and Time Stamp Length specify the number of bits that will be used to convey clock references and time stamps respectively and hence determine for how long timing information can be specified unambiguously. These lengths are chosen according to a set of assumptions and application-dependent requirements. For instance, an application wanting to synchronize streams with a precision of 1 ms (OCR Resolution = 1000) and requiring unambiguous time line for 24 hours can choose the OCR Length accordingly as follows.

$(2^{\text{OCR Length}} / \text{OCR Resolution}) = 24\text{h}$

OCR Length = 27 bits

Time Stamp Length can also be deduced similarly. As mentioned before, there are two types of time stamps defined by MPEG4, DTS and CTS. Both of these time stamps use the same length (OCR Length).

Once the configuration for the SL packets has been determined, they can be constructed based on the data that is to be transmitted. Figure 24 illustrates the structure of the SL packet header according to MPEG4.

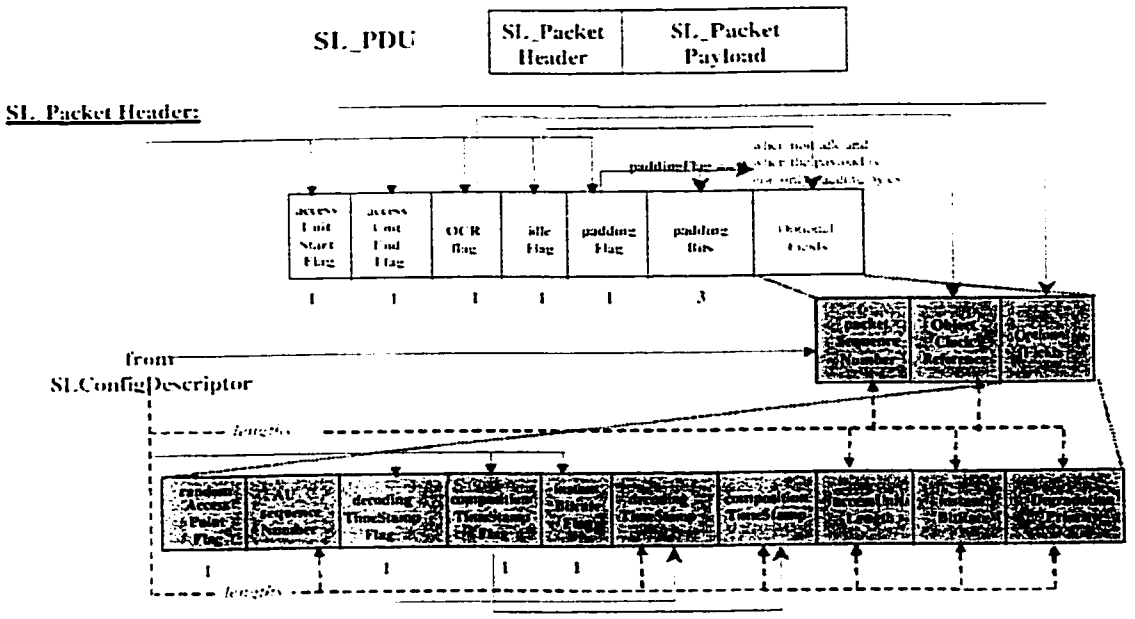


Figure 24. SL Packets

Simply, the OCR flag indicates whether the packet header contains an OCR or not, while decoding time stamp and composition time stamp flags indicate whether their corresponding time stamps are present in the header. If the flags are sent, the SL Packet is known to carry timing information corresponding to the set flag. The number of bits to carry the OCR or the time stamps is obtained from the SL Configuration Descriptor. The various flags allow applications to customize the SL Packets according to their needs and necessities.

4.2 *Implementation of the Synchronization Layer*

In order to implement this layer, Java classes representing the components discussed above were introduced into the framework as well as some interfaces as shown in Figure 25.

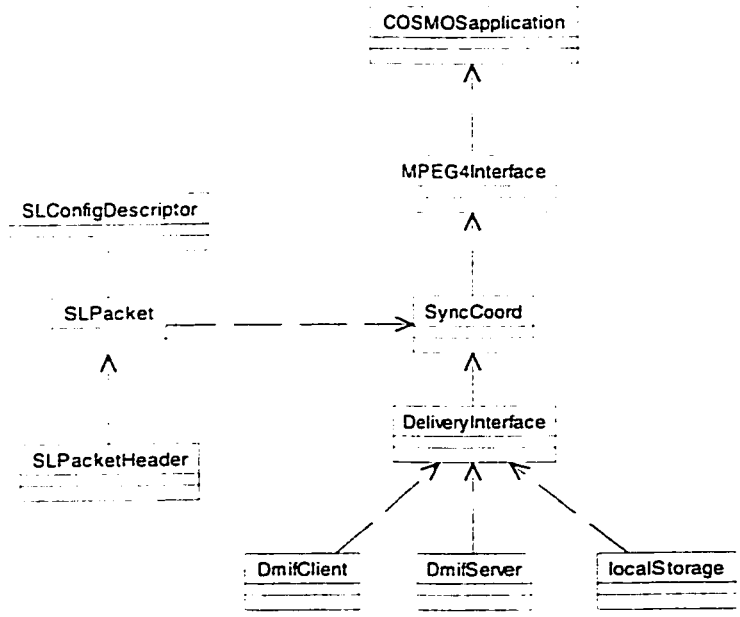


Figure 25. Implementation of the Synchronization Layer

"COSMOS Application" represents a generic application making use of the developed framework. As will be shown later, the recorder and the player will be specific instances of this class. The "MPEG4 Interface" provides for the application an interface to the MPEG4 based framework in order to hide its details. It provides common capabilities such as joining and leaving a multicast session, loading scenes or parts there of from the local directory, sending messages to the multicast group and receiving data from it. These common capabilities may be used by any collaborative application regardless of its specifics.

The "Sync Coord" is in essence the elementary stream interface (ESI) introduced before and hence generates SL packets and exchanges them with the delivery layer.

The interface to the delivery layer (DAI) hides the details of transmission from the higher levels and only provides generic capabilities such as joining and leaving a session and

sending and receiving data. This abstraction allows the multiplexing of SL packets from different sources. For instance during a collaborative session, the application can receive scene updates from one source while receiving from another source a video stream pertaining to a visual object in the scene.

The implementation of MPEG4's Synchronization Layer as part of a generic framework can potentially cater to many applications. In tightly collaborative or in other words synchronous collaborative sessions, the efficient conveyance of timing information is of great benefit. Applications running on networks with delay and especially jitters in delay can also benefit from a specification of temporal characteristics among elementary streams in order to maintain synchronicity. There are many more applications to be sited, two of which were implemented in order to use and showcase the synchronization layer of the framework. The first is recording and playback of collaborative sessions and the other synchronizing animations.

4.3 Recording of a Collaborative Session

Recording a media session involves capturing events, assigning temporal information and storage of the result. The framework allows applications to join multicast sessions as data producers and/or data consumers. Data consumers are informed of updates produced by data producers. Therefore a recorder of a session needs only to be a consumer of data in order to have access to the events relating to the session as shown in Figure 26.

The approach of the recorder joining the collaborative session as another application, called "non-obstructive recording", has the advantage of leaving the application unaware of the recording and hence requiring no change in the behavior of the original or even generating applications [35]. By joining as a consumer, the recorder is informed of any updates produced by any one of the collaborators and hence can proceed to record the sequence of events. This approach is limited to virtual environments where every participant is informed of all events. A large-scale multi-user virtual environment however, can be composed of many "locales" where participants in different locales may not be aware of events occurring in other locales [4]. The "non-obstructive recording"

method would then only be able to record a subset of events happening in the entire virtual environment. In such a case, the recording mechanism would have to be extended.

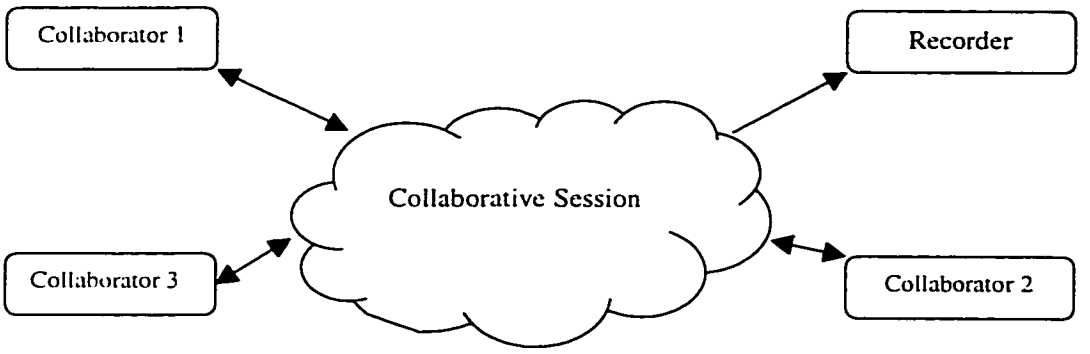


Figure 26. Recorder Joining as a Consumer

Having received the relevant events, the second step in the recording process is deducing the temporal relationship between events and updates. This is done simply as follows:

1- The time of the receipt of the first event from the session is taken to be the starting point of the initial clock reference of the recording and is treated as an Object Time Base (OTB).

2- Any subsequent event received is associated with a time stamp corresponding to the difference in time of receipt of the event and the initial reference obtained in step 1. This time is treated as a Decoding Time Stamp since that is the time when the event reached the decoder of the application.

It should be mentioned that assigning temporal information to events based on the time they were received (as opposed to the time they were generated) results in recordings that are not completely faithful to the original sequence of events if network delay jitter is present. Figure 27 illustrates the recording process and the possible inconsistency as a result of delay jitter. The time between Event1 and Event2 occurring at the generator's station ($T_1 - T_0$) is less than the time between the same events on the receiver station ($t_1 - t_0$), resulting in the recordings being different than the original events. The inconsistency mentioned can be regarded as a disadvantage of the "non-obstructive" method of recording.

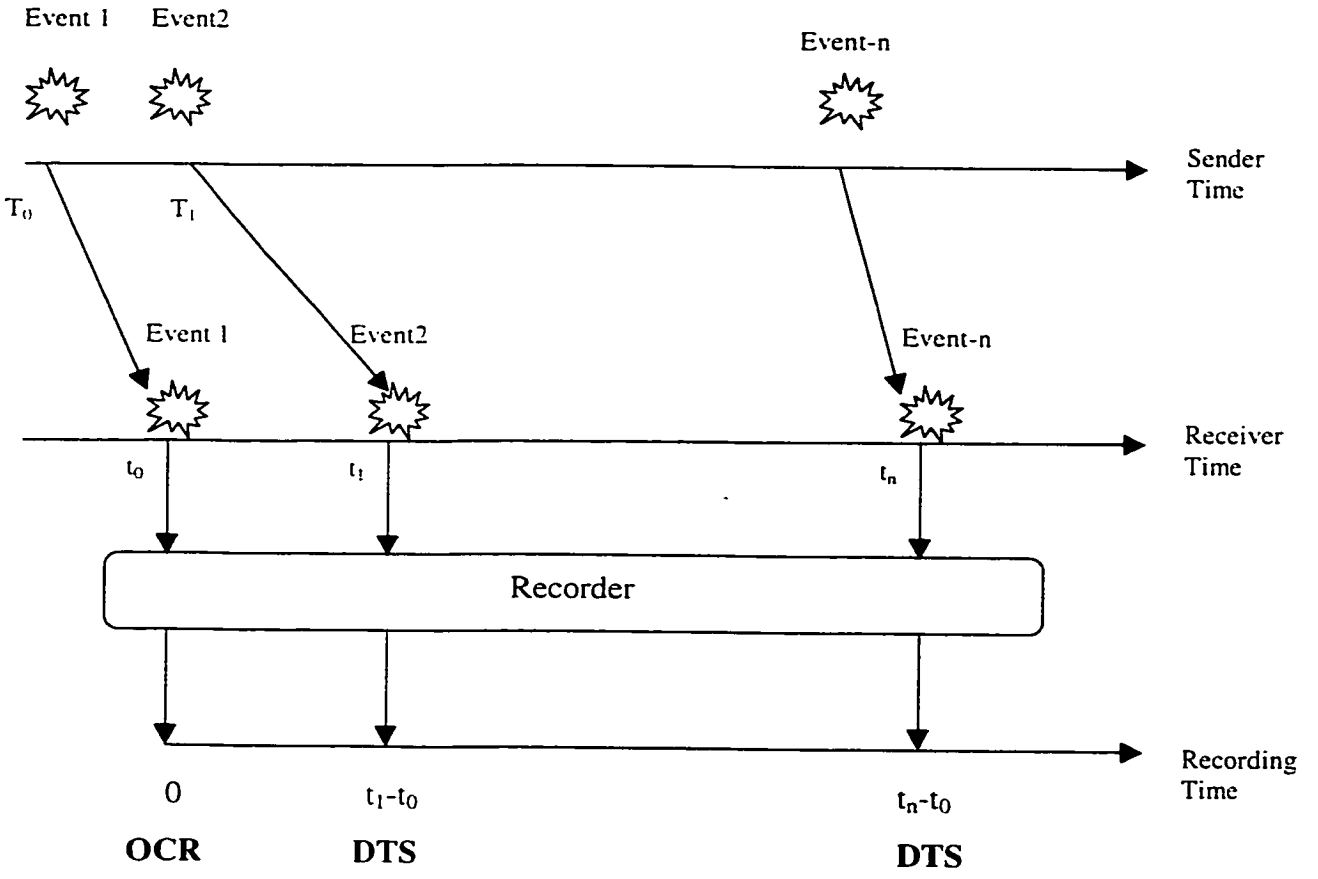


Figure 27. Recording Process and Possible Inconsistency

The above is possibly the most simplistic of recording mechanisms and must be extended to include different scenarios in order to become a comprehensive recording application. However, the aim is to showcase the possible uses of the synchronization layer of the framework and hence a simple recorder suffices for now.

The next step in the process is the storage of the events and the corresponding time stamps. This is done through the framework's synchronization and delivery layer. While the former encodes the time stamps according to MPEG4, the latter stores the resulting SL packets in local storage as shown in Figure 28.

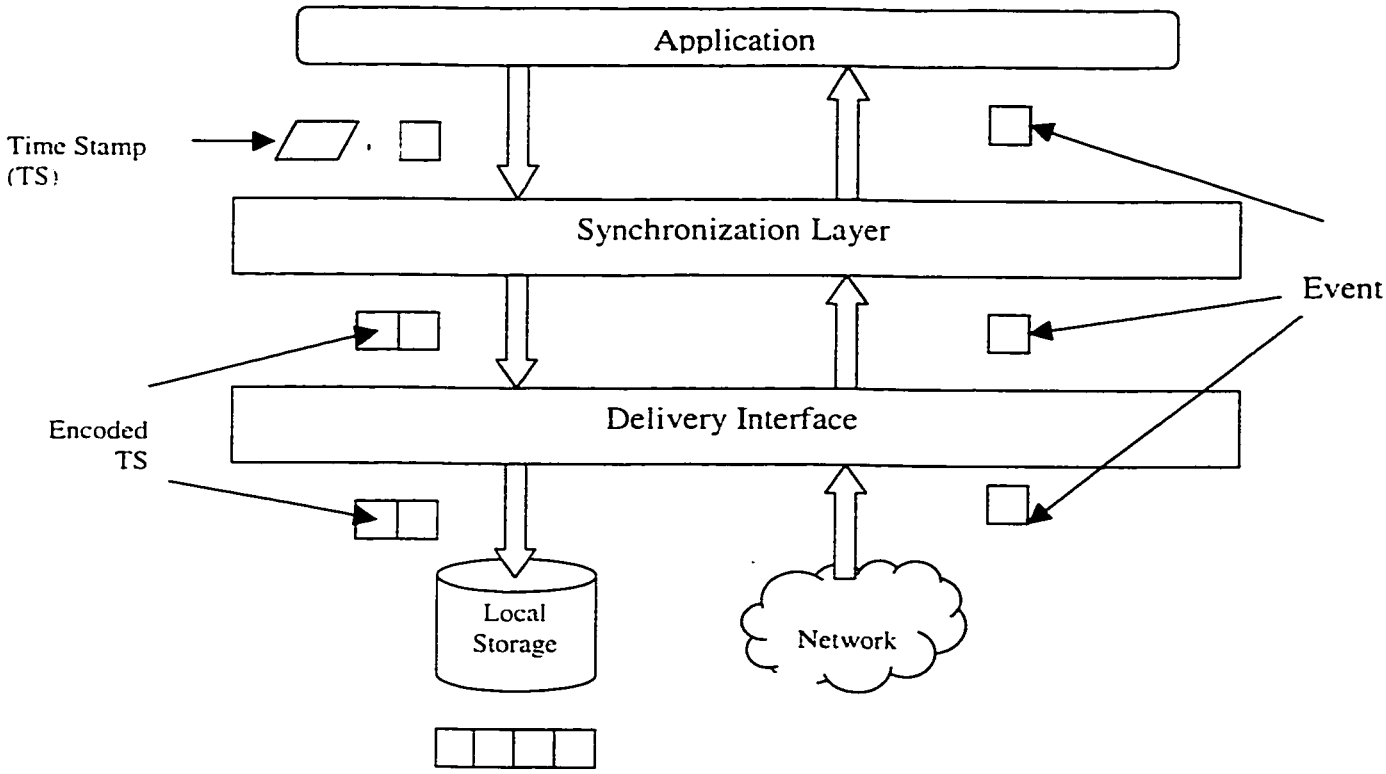


Figure 28. Storage of Recorded Information

As can be seen from the diagram above, the framework is designed such that it does not need to be changed in order to accommodate the recording or, as we shall see later playback. The synchronization layer encodes the time stamps unaware of the destination of the SL packets and the delivery interface sends the data to the URL specified by the application that happens to be a local directory. The framework operates as though taking part in a session and only the application is aware that recording is taking place.

4.4 Playback

The result of the recording process outlined above is a series of updates, in the form of either BIFS-Commands or BIFS-Anims, appended with the corresponding time stamps. To play back the recorded events, the framework decodes the SL packet stream and informs the application of the event and the corresponding time stamp. The delivery interface of the framework abstracts the reception of the SL packet stream and so the synchronization layer need not be modified since the streams are presented in the same

way irrespective of their source. So once again, the framework is kept unaware and functions as though participating in a session. It reads the stream of data from the local directory, passes the data to the synchronization layer that decodes the time stamps, being unaware of the source and how the time stamps will be used. The player application decodes the stream of updates according to the time stamps received, waiting between events according to the interval specified by the decoding time stamps. The result is the playback of the events as they were received during the recording.

In order to playback the recordings at a different rate (slower or faster), the user can specify a rate (1 for normal rate, higher for faster, lower for slower) and this rate is used at time of playback to regulate the occurrence of events. This is accomplished by dividing the waiting time between events by the rate specified. It should also be mentioned that the user is able to change the rate during the playback as well.

Other features such as rewind or random selection of an event during the course of a playback are not yet realized but need to be developed in the case of a more comprehensive player of MPEG4 BIFS sessions.

4.5 Synchronization of Animation

As mentioned in section 3.3.2, an obstacle in the way of sharing animation between participants is the difference in system clocks between different stations. When an animation is triggered on one machine, the trigger time is sent to all other participants but there is usually a difference between the times of different machines. This is yet another area where the implementation of the synchronization layer can have an impact. As show in Figure 29, if there is no synchronization mechanism, the triggering of animation between different machines will take place at different times.

The time on Station A is 10 seconds ahead of Station B. The time of the animation trigger on Station A is 1010 and hence this value is sent to the other station. After the transmission delay, Station B will still wait about 10 seconds before triggering the animation, an incorrect result. Sending time stamps instead however can lead to a more synchronized triggering of animation across different stations.

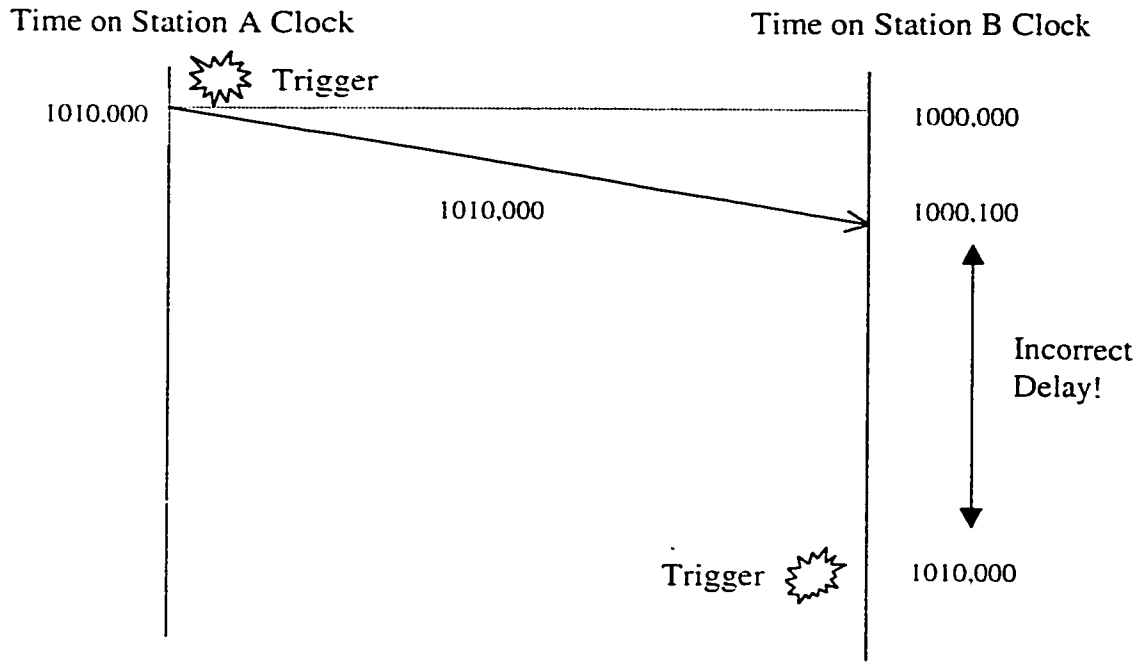


Figure 29. Animation Triggering without Synchronization

An object clock reference accompanies the BIFS packets when the object and its animation are sent to all the participants. From then on, the time of the triggering of animation is transmitted as a composition time stamp as shown in Figure 30.

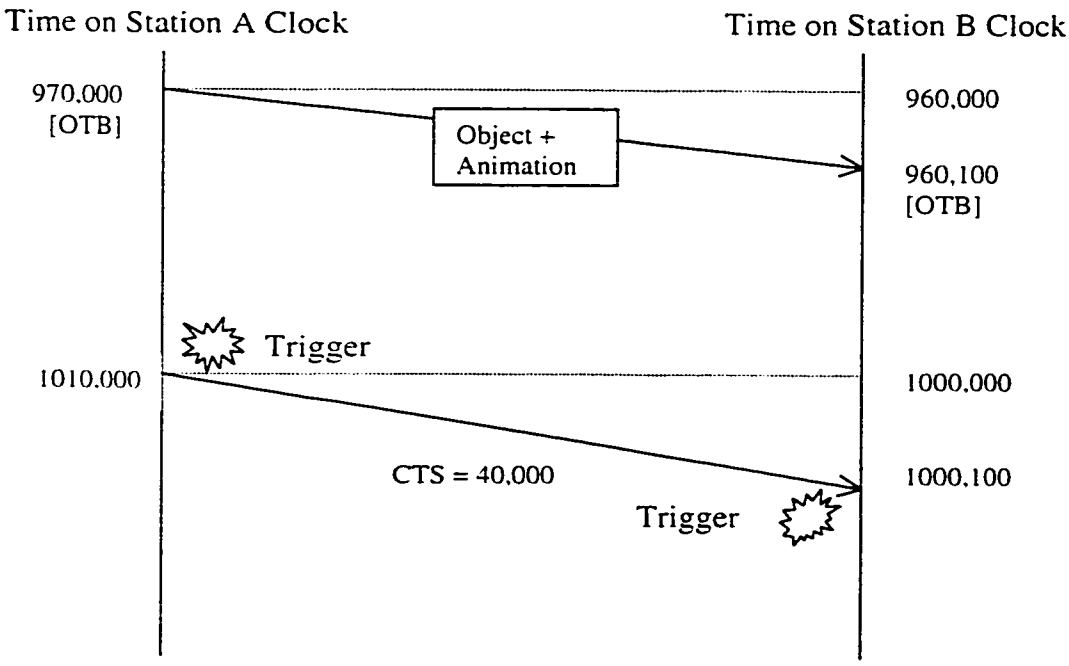


Figure 30. Animation Triggering with Synchronization

Sending the clock reference initially allows the triggering application to send any subsequent events with a time stamp indicating a difference between the current time and the clock reference, therefore eliminating the need to send absolute clock values. Assuming that the transmission delay is constant and uniform, the triggering will happen in synchrony on all receiving stations with a delay (with respect to the triggering of animation at the originator) equal to the network delay.

Chapter 5

5 Development

In order to illustrate the capabilities of the framework discussed, an application prototype is developed for the purpose of a simple training scenario. The prototype is used as a CVE in which a trainer instructs a trainee on how to assemble a bicycle from its parts.

At first, a multicast group is started with two users subsequently joining the session. A generic training environment (a room) is loaded by one of the users and transmitted to the other user in BIFS format. Each user chooses a geometry to represent him/her as an avatar in the environment. Figure 31 shows the view of each user as they see it on their computer.

Figure 31. Two Remote Users Starting the Session

At this point, both users can move around in the virtual room and can see each other's avatar and head movements. Figure 32 shows user B watching user A move across the screen and looking around.

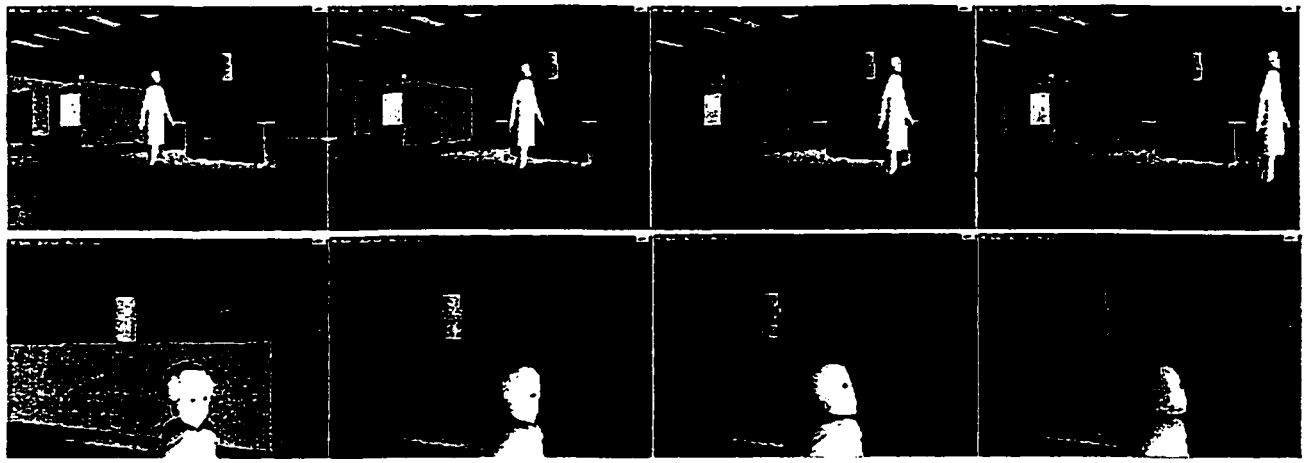


Figure 32. Sharing Avatar and Head Movements

Each user then has the ability to load VRML or BIFS objects or scenes from the local directory and to share them with others. Figure 33 below shows the environment after one user has loaded a bicycle frame into the shared space.

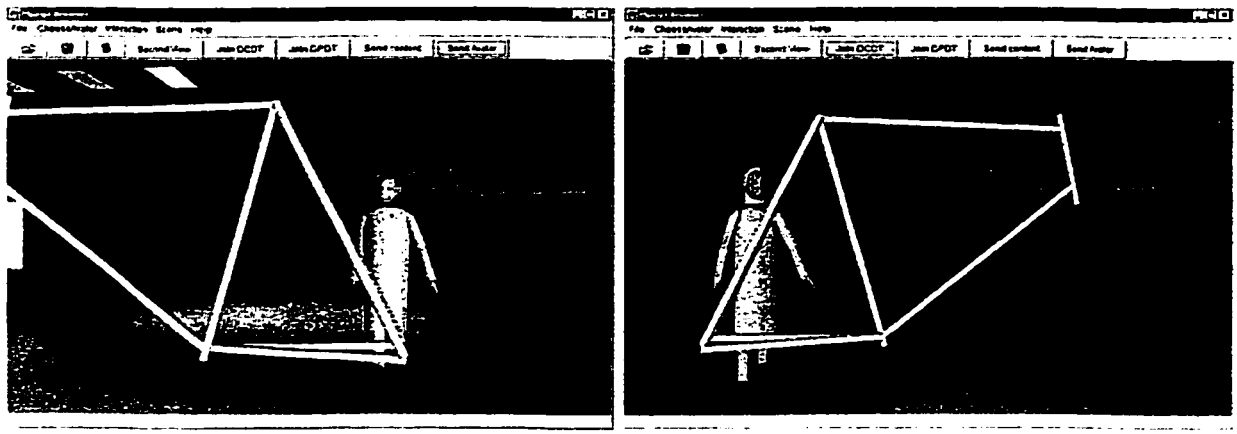


Figure 33. Sharing of Objects from the Local Directory

The users can proceed to load different bicycle parts from their local directory and sharing them with the other participant. They then take turns assembling the bicycle together as shown in Figure 34.

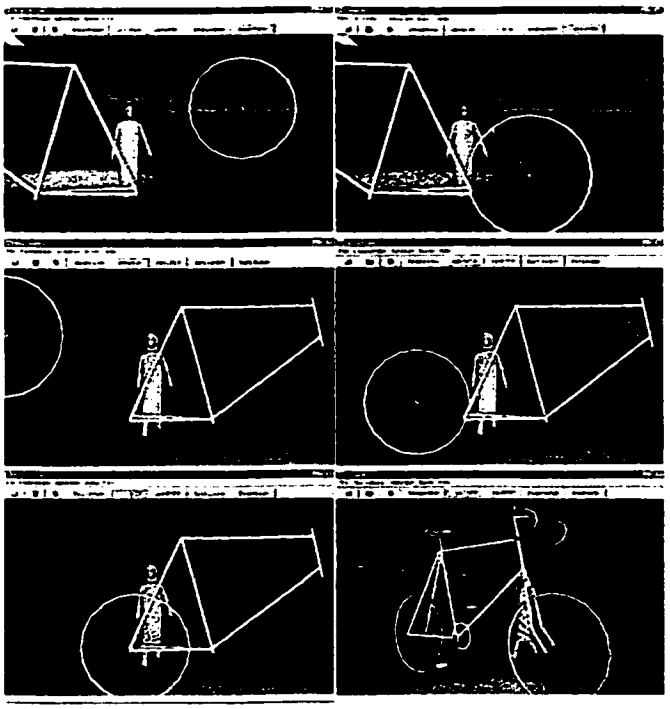


Figure 34. Sharing of Interaction and Collaborative Assembly

Ultimately the bicycle is assembled as a whole and can be saved by each user on the local storage so they could each have a copy of the new scene. Furthermore, the session that took place can be recorded and played back at a later time with the users controlling the speed of the playback (fast forward and slow motion).

If any of the users is in possession of a CyberGlove™ and miniBird™ device, then the movements of the right hand and fingers can be tracked and shared with the other users as well as shown in Figure 35.

The ultimate goal is to use the cyber glove and tracker as the primary user interface with which users manipulate and change the environment around them.

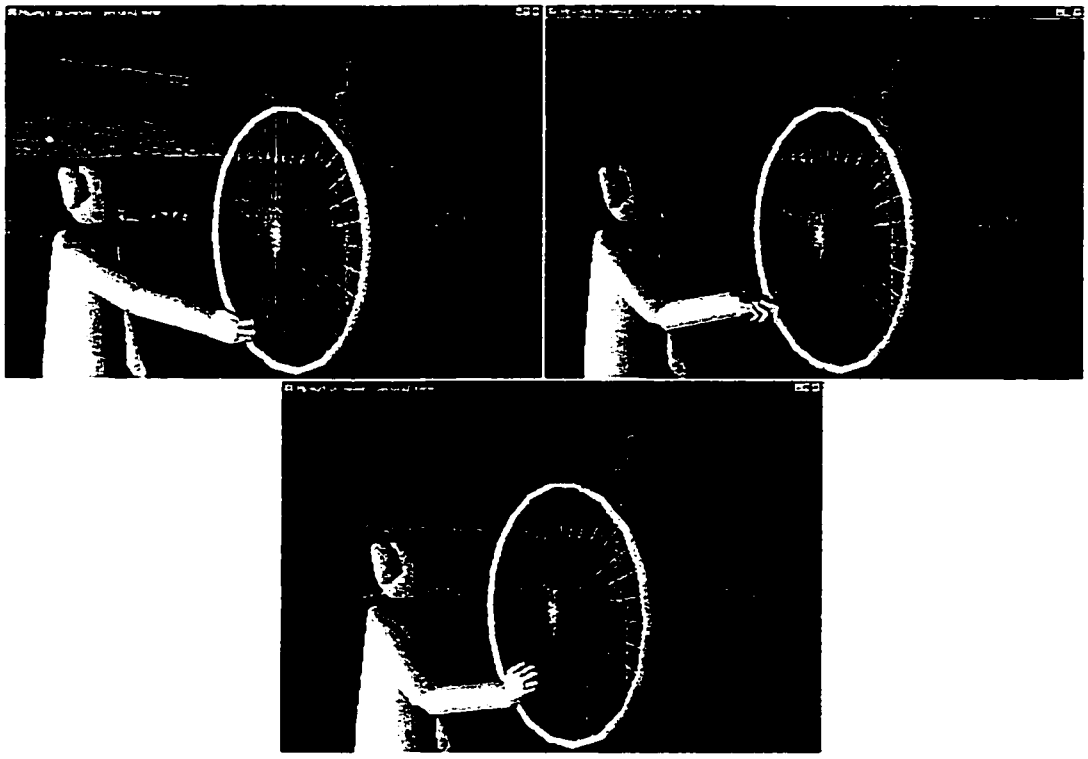


Figure 35. Tracking Hand and Finger Movements

Chapter 6

6 Performance Evaluation

Utilizing an international standard as the core technology of a framework has great advantages with respect to usability and independence from implementation as has been discussed before. It is still of interest however to evaluate the performance of such a framework in order to infer other advantages or possibly disadvantages. There is a number of parameters worthy of study when evaluating a framework designed for the development of CVE applications. The compression ratio of BIFS encoded scenes and objects, effectiveness of BIFS access units and the resulting client-to-client delay are some of these parameters. In addition, the size of a BIFS recorded session can also be an object of evaluation.

6.1 Compression Ratio

BIFS is an encoded scene description and hence can be compared to other formats for scene description. One of the most popular and widely used formats for 3D-scene description is VRML. In fact, BIFS' structure and node classification is designed very similar to VRML, so much so that one can somewhat view BIFS as an encoded VRML. It is therefore interesting to observe the effectiveness of the encoding and compression of BIFS. Table 1 and Table 2 show the comparative size and the compression ratio of BIFS and VRML.

The fields in the BIFS encoded objects shown in Table 1 have not been quantized (Every float occupies 32 bits). As a result, the compression ratio observed is only due to the binary description and context dependency. Nevertheless compared to VRML, BIFS exhibits at least a 2:1 compression ratio but the actual compression ratio depends on the structure of the object.

In order to investigate the effect of BIFS field quantization on the compression ratio, two fields whose quantization was thought to make the most impact were quantized. Geometries are most commonly described using an "IndexedFaceSet" node, already introduced in section 2.4.1 that uses a list of 3D coordinates (described by the "Coordinate" field) whose grouping (described by the "coordIndex" field) specifies the geometry. The more complex the geometry, the longer the list of 3D coordinates and the corresponding indices, creating large arrays of floats and integers. The specifications for BIFS describe a quantization process applicable to many numeric fields in a scene description. The two fields mentioned above ("Coordinate" and "coordIndex") were quantized in order to observe the significance of the quantization with respect to the compression ratio. For all objects, 8-bit quantized fields replaced 32-bit floats and integers. Table 2 summarizes the results.

The first observation is that there is a significant increase in the compression ratio from Table 1 to Table 2. This compression ratio has a potential to increase due to further improvements. Firstly, many fields other than the two quantized for this test can be quantized, further reducing the size of the description and secondly, the quantization process can be adapted to specific objects, hence producing better results.

Comparing individual object compression results of Table 2 to that of Table 1, one can better see the variance of compression ratios due to the difference in object descriptions. For instance, the "Wheel" object that exhibits a low compression ratio in Table 1 has a much higher ratio in Table 2, signifying that this object is primarily described in terms of an "IndexedFaceSet" node.

Independent of the structure of a scene, one can expect a minimum compression of 4:1 from BIFS with respect to VRML. Coupled with the fact that unlike VRML, BIFS encoded scenes can be streamed, the compression of scenes in BIFS makes this standard description all the more attractive for use in CVE applications.

Table 1. BIFS Compression Ratio

3D Object	VRML Text(bytes)	BIFS (bytes)	Compression Ratio
Box	307	58	5.3:1
Table	8817	3091	2.9:1

Room	22385	7940	2.8:1
Wheel with animation	40718	24353	1.7:1
Hand	76155	38101	2:1
Bicycle Frame	77440	36315	2.1:1
Gear	84007	39912	2.1:1
Avatar1	119444	47665	2.5:1
ATM Switch	267246	102551	2.6:1
Avatar2	196689	81794	2.4:1
City	5608874	1268811	4.4:1

Table 2. Quantized BIFS Compression Ratio

3D Object	VRML Text(bytes)	Quantized BIFS (bytes)	Compression Ratio
Box	307	58	5.3:1
Table	8817	2116	4.2:1
Room	22385	4678	4.8:1
Wheel with animation	40718	7229	5.6:1
Hand	76155	10431	7.3:1
Bicycle Frame	77440	9794	7.9:1
Gear	84007	11211	7.5:1
Avatar1	119444	27397	4.4:1
ATM Switch	267246	72481	3.7:1
Avatar2	196689	35376	5.6:1
City	5608874	428429	13.1:1

6.2 *Encoding/Decoding Time*

One of the ways that the encoding of media is beneficial is that an object smaller in size will be transmitted in a shorter period of time. It is of interest however to know how much time is consumed by the encoding and more importantly the decoding process. Table 3 summarizes the encoding and decoding time of some objects. The objects were encoded/decoded 10 times on a PentiumII 450MHz PC with Windows NT and the average was noted.

Table 3. BIFS Encoding/Decoding Time

3D Object	Encoding Time (ms)	Decoding Time (sec)	Total (sec)
Room	270	1.4	1.67
Hand	450	1.6	2.05
Avatar	310	2.1	2.41
City	235000	85	320

As expected, the total of encoding and decoding time increases as size increases. However, it is interesting to note that the decoding time is longer than that of encoding, hence indicating an asymmetrical coding relationship. The encoding time is not of great consequence since objects can be encoded once and used and transmitted subsequently. The decoding process will however play an important part and hence measures should be taken to increase its efficiency. The last entry in the table is of special interest because of the very long decoding time due to the large size of the object. A user can not be expected to wait for 85 seconds for the entire object to be decoded but here is where the streaming of BIFS would make its impact. The application could be modified to display the parts already received and allow the user to navigate and explore the environment as other parts of the environment are received. In this way, the user takes part in the application before the scene in its entirety is received.

Once again, the primary concern with regards to the decoding time is whether any time is saved in transmission of encoded objects. Table 4 lists the average client to client delay (CCD) in transmitting the encoded objects and decoding them on the receiver side. For

the purpose of this particular test, two PentiumII 450MHz running Windows NT were used over two networks (with no intermediate nodes) and of different speed.

Table 4. Object Client to Client Delay

3D object	CCD on 100Mbps Ethernet(sec)	CCD of transmitting BIFS on 38.4kbps serial connection (sec)	Transmission delay of sending VRML on 38.4kbps line (sec)
Room	1.4	3	5
Hand	2.5	9	16
Avatar	2.8	11	25

Comparing the results over the 100Mbps Ethernet to that of Table 1, one can observe that the main source of delay, as expected, is the encoding/decoding time. On a slower line however, the advantage of transmitting encoded data becomes evident since it takes a shorter time to transmit and decode BIFS data than to transmit the VRML file. (Maybe compare to zipped and parsing as well as quantized)

6.3 Update Client to Client Delay

In addition to the encoding and transmission of scene and objects, a CVE application will update the scene on a regular basis whose effectiveness partially relies on the client to client delay of such update messages. The acceptable simulation and network delay in some studies is indicated as 100 ms [37]. In order to measure the client to client delay of an update message, a sample message was encoded and sent by one computer to another. Upon receiving and decoding the update message, the receiver computer would then encode and send the same update message. The time elapsed from the original encoding to receiving and decoding a similar message is twice the delay. The process of passing updates continues for a specified period of time after which the average delay can be calculated. The test was done with two Pentium II 450MHz computers over a 100Mbps link with no intermediate nodes. The update message used was a change in translation, which encoded in BIFS occupies 17 bytes. The resulting delay was measured as 51ms, a

value below that of the acceptable level. In addition, comparing this value to those obtained in a study of collaborative technologies [32] under the same conditions illustrates the effectiveness of using BIFS updates as shown in Table 5.

Table 5. Update Client to Client Delay

	RTI with VRML	RTI with Java3D	Spline with VRML (100ms update)	Spline with VRML (1ms update)	COSMOS
CCD	184	152	129	68	51

It should be mentioned that a primary source of delay in the COSMOS framework was found to be the JMF RTP buffering mechanism. A more efficient transmission scheme would help reduce the client to client delay of the system.

6.4 Recording Size

The size of a recorded session is highly dependent on many factors such as the size of the objects used, number of participants, number of updates generated by each participant and how much quantization was utilized to encode the updates. It is difficult therefore to compare a recorded session with another but a general impression can be formed about how large the resulting file size will be in the case of a recorded session. For instance, the recording of a 3 minute long session involving one participant in a room assembling a bicycle from its parts was found to generate a 237KB file, half of which contains the geometries and the rest the updates.

Chapter 7

7 Conclusion

The development of Collaborative Virtual Environment applications requires study and implementation of a wide variety of issues and technologies one of which is efficient representation of 3D scenes and their corresponding update. The MPEG4 standard addresses this issue in the form of BIFS, making it possible to construct a framework for CVE applications based on an internationally accepted standard. Once the framework has reached the level of maturity where a reasonably complex collaborative session can be carried out, the possibility of recording such sessions comes to light. Similar to the recording of audio/video, recording of virtual sessions can serve as a way of media production as well as fulfilling other applications. In accordance with the idea of using international standards for the development of the framework, MPEG4's Synchronization Layer is utilized to represent and encode temporal information related to a session. This information can in turn be used to record a session and at a later time be played back.

This thesis presented the extension of COSMOS from an emerging framework to a more mature framework supporting various interactions and features as well as the ability to incorporate temporal characteristics to objects and streams. There are however a great many issues yet to be investigated with regards to CVEs, leaving many paths of improvement and inviting further research. By treading such paths by way of future studies, the COSMOS framework has the potential to become a leading technology in the development of CVE applications.

Chapter 8

8 Issues and Future Work

The design of any architecture or framework, its implementation and the development of an application give rise to many difficulties and issues worthy of mention as well as the possibility for further improvement. The following is a list of issues encountered and potential enhancement to the framework or the application:

8.1 VR Realism

One area where any virtual reality application can always enjoy more improvement is the addition of features that make the application more realistic for its users. Some of these features more specific to the application developed in this thesis are listed below.

The limited computer human interface to the virtual environment is usually a significant handicap whilst only a mouse or keyboard are used for this purpose. The integration of a cyber glove and tracker into the interface is only the first step towards a more realistic environment for the user since these input devices are yet not used to interact with the scene. In order to become useful, the glove and tracker interfaces must be used at least to move objects within the scene. Otherwise they serve no useful purpose.

The next step in improving the human computer interface is to incorporate force feedback mechanisms into the system. Unless the user can feel the objects being manipulated, the control over the object is difficult and unrealistic. In order to provide force feedback, collision between objects must be detected accurately and be used in conjunction with feedback devices.

For a virtual environment to be realistic, it must exhibit physical laws similar to those present in the real world. For instance the inability of solid objects to pass through one another in the real world must be translated into the virtual world in the form of a collision avoidance capability. Additionally without gravitational forces, objects can

remain suspended in air as they are manipulated. More importantly with respect to truly collaborative applications, where two or more users wish to interact with the same object, the physical laws pertaining to force and torque must be incorporated into the system.

At this point, the system allows avatars to move in the scene by this movement does not resemble the walking of humans since the corresponding hand and leg movements are not present. The addition of walk animation to the avatar movement is another way of making the application more natural.

Simulating virtual environments on a desktop computer has many limitations, one of which is the lack of peripheral vision. Having used their peripheral vision in the real world, the users of virtual environments feel their views as too restricted due to a flat display. CAVE [7] or Head Mounted Displays are some of the ways to enhance the virtual experience.

When implementing the mapping of the viewpoint to the head of the avatar, the question of restricted head movements came into light. In the real world, humans can at most rotate their heads 180^0 degrees in either direction. In order to see other objects around them, one moves the upper body or at most the whole of the body. Restricting the movements of the head of the avatar to only 180^0 seems as an uncomfortable restriction in the virtual environment since the user does not have control over the upper body of the avatar. Combining this with the limited peripheral view in the VE discussed above results in a very limited viewpoint. One possible solution is the use of peripheral lenses [16].

As has already been alluded to, the lack of control of the user over the upper body of the avatar causes some limitations. Other than the limited view, there is also a limited reaching capability. Humans can reach with their hand a relatively wide area around them with the movement of not only their hands, shoulders and elbows, but also their hip and upper body movements. Tracking only the arm (hand, shoulder and elbow and not hip and upper body) results in a very limited space where objects can be reached. In order to reach other objects, the user is forced to move the entire body, making the process more cumbersome.

8.2 *Further Enhancements to the COSMOS Framework*

MPEG4 defines encoding specific to face and body parts of an avatar. At this point, the avatar body movements (head, hand and fingers) are encoded as regular BIFS updates and the specific MPEG4 encoding is not used. The implementation of those nodes as a part of the framework will result in greater efficiency of updates pertaining to body movements.

BIFS-Anims are used at this point to convey changes pertaining to the avatar including the movement of the body, hand and fingers. If BIFS-Anims are to be used to update manipulation of objects, the entire BIFS Configuration must be transmitted every time a new object is to be added to the animated list. The MPEG4 standard can be extended to allow update of the BIFS Configuration so that the entire configuration would not have to be sent every time.

As discussed in section 2.4.2 there are some differences between MPEG4 and Java3D in their treatment of animation. While more than one interpolator can be associated with the same node in MPEG4, Java3D only allows one interpolator per node. As a result, MPEG4 animations loaded into COSMOS would not exhibit their intended behavior. In order to remedy this inconsistency, the structure of the MPEG4 file is changed by associating each interpolator with only one node. A better solution is to change the loading process in COSMOS to dynamically perform the necessary changes.

When implementing parts of the MPEG4 standard, there are circumstances where the standard can not be followed exactly. For instance, when using JMF RTP to transmit BIFS data, JMF built-in classes must perform the packetization but this task is to be performed at the ESI according to MPEG4. As a result, the clear interface lines defined in the standard are blurred in the actual implementation.

One of the features lacking in COSMOS is the absence of ownership and an access mechanism. In some CVE applications especially in the area of education and training, it is desired to give one user (teacher or trainer) exclusive access to certain objects in order to disallow the intervention of other users. This feature and more generally providing an access model [33] are necessary future improvements of the COSMOS framework.

Providing an access model is only applicable to certain applications however. In some other applications, mutually exclusive ownership of one user over an object is

undesirable and instead two or more users are required to interact with the same object. For such applications, a more general collaboration framework must be developed that allows the simultaneous interaction with an object by a number of users.

As mentioned in section 2.1.2.1, quantization parameters can be specified for individual elements or objects in the scene in order to customize the quantization for the benefit of better compression. However, the implementation of such an adaptive quantization process requires the application to have knowledge of the nature of data contained within the element to be quantized. For instance, in order to assign proper quantization parameters for an IndexedFaceSet node, the application must specify the minimum and maximum values of its Coordinate field, information that can only be obtained by searching through the array of 3D coordinates. At this point in the framework, a set of global quantization parameters is applied to all values resulting in a simple implementation at the expense of optimal compression.

8.3 Further Enhancements to the Application

The recording and playback of the scene is at this point only a prototype showcasing the capabilities of the synchronization layer of the system. The implementation of rewind, random selection of events in a stream and other post-production features would be steps towards making a more comprehensive application of the synchronization capabilities of the framework. In addition, by utilizing the concept of "temporal links" [12], the recording and playback of collaborative sessions can be used more flexibly. For instance, a recording of a past session can be played inside a current collaborative session and the resulting session can in turn be recorded as well.

As mentioned before, the synchronization layer implemented is general and not specific for one particular application and so it can be used not only to synchronize synthetic media events but can be applied to natural (audio/video) media as well or a combination thereof. The start, stop and pausing of a video inside a 3D scene can be synchronized between various stations the same way the start/stop of animation was done in synchrony. The inconsistency between what a generator of a stream of events observes and that of the receiver of such a stream is also an issue in need of improvement. In the case of

manipulating an object for instance, the manipulator sees a continuous update of the object but in order to conserve network bandwidth, only a portion of these updates is sent to other participants. As a result, the receivers of such updates do not see a continuous stream of updates but a series of discrete events. One remedy to this problem is the generation of "in-between" values for discrete events to give the impression of a continuous stream.

As has already been mentioned, BIFS scenes can be streamed and therefore the receiving terminal need not wait for the entire scene to be downloaded in order to render parts of it. This capability can best be exploited in the case of large virtual scenes where it might take a long interval for the entire scene to download. In such a case, the receiving terminal can be configured to render the parts received and allow the user to navigate in the scene as the remaining parts are received, decoded and appended to the scene. If the initial part of the streamed scene is the part most proximate to the user, it is possible for the user to be unaware of the incompleteness of the scene because he/she can not see the parts of the scene that have not been received and rendered yet. The ability to render parts of scenes as they are streamed would however come at the expense of rendering efficiency.

8.4 Concurrency Issues

When implementing the playback mechanism of the recorded session, a need for concurrency arose. Since the main thread of execution was made responsible for the loading of the recorded file, the playback was not interactive. In other words, as the session was played back, the user could not navigate and change the point of view as a result of the main thread being occupied with the task of loading. In order to solve this problem and allow the user to interact with the virtual environment as the session is replayed, a separate thread had to be used for the loading, leaving the main thread to attend to user interactions.

When sending the updates in hand and finger movements, another issue related to concurrency had to be addressed. Since both input devices (glove and tracker) had their own thread of control, they competed with one another for access to the delivery layer.

As a result, when the tracker would send updates, the glove could not and vice versa, making the update out of synchrony. In order to solve this problem and reduce the overhead of sending updates independently, the tracker and glove threads were merged into one with the resulting update messages including changes in both the hand and the fingers.

8.5 Future Studies

The issue of scalability is an important parameter worthy of attention and investigation. The effects of the increase in number of users, number of objects, complexity and size of the virtual scene and number of updates on the performance of the application and solution to possible shortcomings can all be topics of future research.

So far, most CVE applications have been developed and used over local area networks where the bandwidth is plenty and reliability not an issue. The feasibility of CVE applications running on the Internet and the resulting issues are of great interest and can pave the way for the further availability and use of CVEs. For instance, error resilience techniques for MPEG4 video streams have already been suggested [13]. It would be of interest to see if such techniques could be applied to BIFS streams as well, hence allowing their transport over unreliable links.

A related matter is that of providing Quality of Service mechanisms specific to CVE applications. Such a mechanism would help users with different resources collaborate with one another using the same system.

The synchronization layer developed works well in environments where the delay is small and constant. Removing such assumptions gives rise to more complex issues related to synchronization, which can in turn be focus of research initiatives in the future.

Bibliography

[1] Angel, E., "Interactive Computer Graphics: a top-down approach with OpenGL", Addison Wesley Longman Inc., p. 314, 1997.

[2] Ascension Technologies Inc, http://www.ascension-tech.com/products/minibird/

[3] Babski, C., Standard Bodies, http://ligwww.epfl.ch/~babski/StandardBody/

[4] Barrus, J.W., Waters, R.C. and Anderson, D.B., "Locales: Supporting Large Multi-user Virtual Environments", IEEE Computer Graphics and Applications, November 1996, pp. 50-57

[5] Benford, S. D. and Mariani, J. M., "Virtual Environments for data Sharing and Visualisation - Populated Information Terrains", Proc. Interfaces to Database Systems (IDS) '94, Lancaster, UK, 1994.

[6] Benford, S., Greenhalgh, C., Snowdon, D., Bullock, A., "Staging a Public Poetry Performance in Collaborative Virtual Environments", Proc. 5th European Conference on Computer Supported Cooperative Work (ECSCW'97), Lancaster, UK, September 1997.

[7] Cruz-Neira, C., Sandin, D.J. and DeFanti, T., "Surround-screen projection-based virtual reality: The design and implementation of the CAVE", Computer Graphics (Proceedings of SIGGRAPH '93), pages 135—142, ACM SIGGRAPH, August 1993.

[8] Darlagiannis, V., "COSMOS: COllaborative System framework based on MPEG-4 Objects and Streams", Masters Thesis, University of Ottawa, Canada, 1999.

[9] Darlagiannis, V. Georganas, N.D., "Virtual Collaboration and Media Sharing using COSMOS." Proc. 4th WORLD MULTICONFERENCE on Circuits, Systems, Communications & Computers (CSCC 2000), Greece, July 2000

[10] Emmen, A., Versweyveld, L., John, N.W., "WebSET: Integrated XML and VR components for collaborative medical training", MEDNET 2000, 23-26 November, Brussels, Belgium.

[11] Greenhalgh, C., Benford, S., "MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading", Proceedings of the 15th International Conference on Distributed Computing Systems (DCS'95), Vancouver, Canada, June, 1995.

[12] Greenhalgh, C., Purbrick, J., Benford, S., Craven, M., Drozd, A. and Taylor, I., "Temporal Links: Recording and Replaying Virtual Environments", Proceedings of the 8th ACM international conference on Multimedia (MM 2000), pp. 67-74, 2000, ACM Press.

[13] Gringeri, S., Egorov, R., Shauaib, K., Lewis, A. and Basch, B., "Robust Compression and Transmission of MPEG-4 Video", ACM Multimedia 99, November 1999, pp. 113-120

[14] Guye-Vuillieme, A., Capin, T.K., Pandzic, I.S., Thalmann, N.M., Thalmann, D., "Non-verbal Communication Interface for Collaborative Virtual Environments", The Virtual Reality Journal, 1999 Vol. 4, pp.49-59.

[15] Hill, F.S., "Computer Graphics", Macmillan Publishing Company, p. 527, 1990.

[16] Hindmarsh, J., Fraser, M., Heath, C., Benford, S. and Greenhalgh, C., "Object-Focused Interaction in Collaborative Virtual Environments, in ACM Transactions on Computer-Human Interaction (ACM TOCHI) Special Issue on Collaborative Virtual Environments, December 2000, ACM Press

[17] Human Animation Working Group, http://www.h-anim.org/

[18] ISO/IEC 11172, "Coding of Moving Pictures and Audio", 1993

[19] ISO/IEC 13818, "Generic Coding of Moving Pictures and Associated Audio Information", 1998

[20] ISO/IEC 14496, "Coding of Audio-Visual Objects", 1999

[21] ISO/IEC 14496-1: "Information Technology: Coding of Audio-Visual Objects. Part 1: Systems", p. 83, 1999

[22] ISO/IEC 14496-6: "Information Technology: Coding of Audio-Visual Objects. Part 6: DMIF", 1999

[23] ISO/IEC 14772: "Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML)", 1997

[24] Java3D Programming Interface, Sun Microsystems, http://java.sun.com/products/java-media/3D/

[25] Java Media Framework, Sun Microsystems, http://www.javasoft.com/products/java-media/jmf/index.html

[26] Java Native Interface, Sun Microsystems, http://java.sun.com/j2se/1.3/docs/guide/jni/index.html

[27] Leung, W.H., Goudeaux, K., Panichpapiboon, S., Wang, S.B. and Chen, T., "Networked Intelligent Collaborative Environment (NetICE)", IEEE Intl. Conference on Multimedia and Expo., New York, July 2000

[28] Liang, J., Green, M., "Geometric Modeling Using Six Degrees of Freedom Input Devices", Proceedings 3rd International Conference on CAD and Computer Graphics, pp. 217-222, Beijing, China, August 1993.

[29] Macedonia, M.R., Zyda, M.J., Pratt, D.R., Barham, P.T. and Zeswitz, S., "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments", in Presence, Vol. 3, No. 4, Fall 1994, pp. 265-287, MIT Press.

[30] McKerrow, P., "Introduction to Robotics", Addison Wesley, 1991.

[31] Oliveira, J.C., Shirmohammadi, S. and Georganas, N.D., "A Collaborative Virtual Environment for Industrial Training", Proc. IEEE Virtual reality 2000 (VR'2000), New Brunswick, NJ, March 2000

[32] Oliveira, J.C., Shirmohammadi S. and Georganas, N.D., "Distributed Virtual Environment Standards: A Performance Evaluation", Proc. IEEE/ACM Third International Workshop on Distributed Interactive Simulation and Real Time Applications (D I S - R T ' 99), Greenbelt MD, Oct.1999

[33] Pettifer S., Marsh J., "A Collaborative Access Model for Shared Virtual Environments", Proceedings IEEE WetIce2001, MIT, Cambridge, USA, June 2001.

[34] Pratt, D. R., Pratt, S. M., Barham, P. T., Barker, R. E., Waldrop, M. S., Ehlert, J. F., and Chrislip, C. A. "Humans in Large-scale, Networked Virtual Environments", Presence, MIT Press, Vol 6, No. 5, 1997, pp. 547-564

[35] Shirmohammadi, S., Ding, L. and Georganas, N.D., "An Approach for Recording Multimedia Collaborative Sessions: Design and Implementation", Journal of Multimedia Tools and Applications (accepted to appear)

[36] Virtual Technologies Inc., http://www.virtex.com/products/hw_products/cyberglove.html

[37] Wloka, M.M., "Lag in Multiprocessor VR", Presence: Teleoperators and Virtual Environments (MIT Press), Vol. 4, No. 1, Spring 1995.

[38] Xj3D Open Source Browser, http://www.web3d.org/TaskGroups/source/xj3d.html