Suitability of Java for Virtual Collaboration

Vasilios Darlagiannis[1], Ralf Ackermann[1], Abdulmotaleb El Saddik[1], Nicolas Georganas[2], Ralf Steinmetz[1],[3]

[1]Industrial Process and System Communications, Dept. of Electrical Eng. & Information Technology, Darmstadt University of Technology, Darmstadt, Germany

[2]Multimedia Communications Research Laboratory, School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada

[3]GMD IPSI, German National Research Center for Information Technology, Darmstadt, Germany

Abstract: Collaborative Virtual Environments (CVEs) support the co-existence of many users in shared virtual worlds, permitting a productive collaboration among them. Due to the inherit complexity of their distributed nature and their big demands on 3D content modeling, CVEs are hard to be developed. The maintenance of CVEs, integrated with naturally captured media streams like audio and video, is an even more difficult and complicated task.
This paper describes how to use Java tools and extensions, like the Java Media Framework (JMF), Java3D and JavaCC, to rapidly develop prototype systems for collaborative virtual environments. It shows our extension of the JMF functionality to support integration with MPEG-4 technology. Our experiences described in this paper have been gained during the development of COSMOS[1], an MPEG-4 based framework for decentralized virtual collaboration.

Keywords: Java, JMF, Java3D, JavaCC, MPEG-4, VRML, RTP, Collaborative Virtual Environments

1 Introduction

1.1 Motivation

Collaborative Virtual Environments (CVEs) are hard to be developed, due to the inherit complexity of their distributed nature and their big demands on 3D content modeling. The maintenance of collaborative virtual environments, integrated with naturally captured media streams like audio and video, is an even more complicated task.
CVEs development needs to fulfill a set of requirements in order to provide an acceptable and usable product. A list of typical challenges is shortly given below:

- Interactivity,
- Scalability,
- Robustness,
- Session management,
- Multicasting,

[1] COSMOS: COllaborative System based on MPEG-4 Objects and Streams

- Data consistency,
- Media encoding,
- 3D content modeling and
- Fast rendering, integrating media streams into the 3D content.

In addition, although fast development of a system is not always a requirement, it is definitely a desired goal.

Obviously the implementation and the integration of the above requirements are highly time-consuming and complicated tasks. Thus, at least a partly support by the programming language and additional tools is definitely desirable. Such tools will be shown in the following section.

Java, the state-of-the-art programming language, provides a set of tools and extensions to its core platform, like Java Media Framework (JMF), Java 3D and JavaCC, to enable the development of advanced applications and systems.

The purpose of this paper is to describe how to use Java extensions and tools to rapidly develop prototype systems for CVEs. In addition, since the proposed ideas have already been implemented in COSMOS [2] - an MPEG-4 [7] based framework for decentralized virtual collaboration - we refer to MPEG-4 concepts to justify design goals and solutions.

1.2 Base Programming Components

The Java3D API [15] is used for writing stand-alone 3D graphics applications or Web-based 3D applets. It offers developers high-level constructs for creating and manipulating 3D geometry and tools for describing the structures used in rendering that geometry. Additionally, it compromises basic functionality for defining objects behavior, animation, as well as user interaction with a scene. Application developers can easily describe very large virtual worlds, which, in turn, are efficiently rendered by the Java3D rendering engine.

The JMF API [14] enables the display, capture, encoding, decoding and streaming of multimedia data in Java technology-based applications and applets. As a standard extension to the Java platform, it delivers the ease-of-development and cross-platform benefits of the Java programming environment incorporating media data such as audio and video into their applets and applications. The API specifies a unified architecture, messaging protocol and programming interface for playback, capture and conferencing of compressed streaming and stored timed media including audio and video across all Java compatible platforms. In addition, JMF provides an API for the RTP (Real-time Transmission Protocol) transport protocol, which is appropriate for transmission of real time media.

Java Compiler Compiler (JavaCC) [16] is a Java parser generator written in Java. It takes a language described in a special format as input and it produces pure Java code. JavaCC generates top-down (recursive descent) parsers as opposed to the bottom-up ones generated by YACC[2]-like tools. This allows the use of more general grammars.

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

2 Content Modeling and Event Handling

VRML [8] is still the most popular format used to describe the 3D content of a virtual environment. Recently, the acceptance of MPEG-4 BIFS (BInary Format for Scene) [7] has been increased, since it allows a more powerful set of functionality on the 3D content. Two of the most important improvements are the ability to compress the size of the data needed to describe the environment, as well as to stream the content and its updates.

Having the ability to handle content described in any of the above formats (VRML and MPEG-4 BIFS) is considered to be very useful for a system. The Java3D API provides the concept of a loader, to achieve the "translation" of the content, described in a suitable format for storage or streaming, into Java3D objects and to enable their rendering. Although a number of VRML loaders are publicly available, there is no MPEG-4 BIFS one, yet. COSMOS [2] provides a first prototype, which is not yet publicly available though.

Because of the many possibilities the content can be described, and the rendering can be achieved (using e.g. other means than the Java3D API), a loose coupling between the internal modeling structure, the Java3D API and the loaders has been agreed upon in our concept. Fig. 1 shows the proposed architecture for achieving that goal.

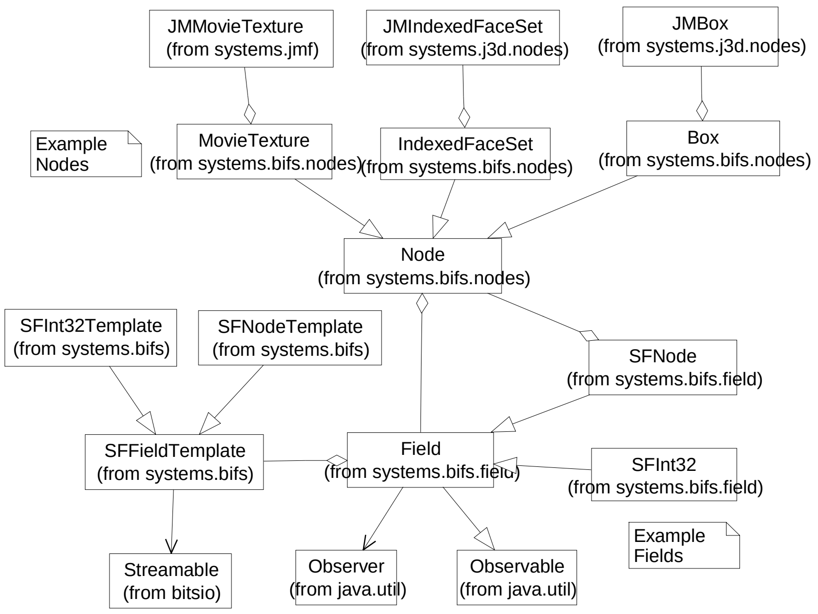
JMMovieTexture (from systems.jmf)
JMIndexedFaceSet (from systems.j3d.nodes)
JMBox (from systems.j3d.nodes)
Example Nodes
MovieTexture (from systems.bifs.nodes)
IndexedFaceSet (from systems.bifs.nodes)
Box (from systems.bifs.nodes)
Node (from systems.bifs.nodes)
SFInt32Template (from systems.bifs)
SFNodeTemplate (from systems.bifs)
SFNode (from systems.bifs.field)
SFFieldTemplate (from systems.bifs)
Field (from systems.bifs.field)
SFInt32 (from systems.bifs.field)
Example Fields
Streamable (from bitsio)
Observer (from java.util)
Observable (from java.util)

Fig. 1. Content modeling architecture

A basic object in the architecture is the Node, which is a base abstract class that models the notion of a VRML or MPEG-4 BIFS node. Example derived classes are the MovieTexture, IndexedFaceSet or Box. Those classes encapsulate the Java3D and JMF related objects, which are the JMMovieTexture, JMIndexedFaceSet or JMBox, respectively[3]. Those objects hide Java3D and JMF details. The Node object has a set of fields, depending on the specific class of the derived node, which are modeled with the Field base class. Instances of the Field abstract class are, for example, the SFNode and SFInt32. While those classes are designed to hold information for the internal model structure, they are not appropriate to encode or decode the information, for example, in the MPEG-4 BIFS format. For this operation, there is another abstract class, called SFFieldTemplate, which implements the Streamable interface. Derived classes are the SFNodeTemplate to encode/decode nodes, the SFInt32Template for the SFInt32 fields, etc. Java reflection is used to handle the big number of nodes and fields, in an efficient way.

As it has already been mentioned, VRML is currently the most popular format used for describing 3D content. VRML is text-based, in contrast to MPEG-4 BIFS, which uses a binary encoding format. For this reason, a parser is required to extract the information from the text description of the VRML file into Java3D objects, which are appropriate for rendering. JavaCC has been used for the implementation of the VRML parser. The specifications of the VRML parser has been described in special format, appropriate for the JavaCC. This format permits the inclusion of Java functions and statements into the rules of the parser. JavaCC translates the mixed set of parser rules and Java statements into pure Java code. This way, it is a straightforward procedure to integrate the parser with the rest of the system. The parser is associated with the VRML loader, a class derived from the abstract Java3D loader.

Both VRML and MPEG-4 BIFS use the notion of ROUTEs to describe events in the virtual environments. A ROUTE is an association between two fields of the same type. The first field of the association is the source of the event and the second is the destination. Whenever the value of the first changes, the second is updated accordingly. The observer pattern [4] is fitting perfectly in the implementation of the ROUTEs. Every field can be both the Observable object, as well as the implementation of the Observer interface. Whenever a ROUTE exists, the destination field is being added in the list of the Observers of the source field. This way, a straightforward implementation is possible.

3 Extending JMF

Currently, the implemented JMF RTP supports transmission of audio and video streams only. But, in the development of CVEs, user interactivity transmission is of the highest importance to enable collaboration. User interactivity, like navigation into the shared world, can be encoded in BIFS-command streams. Moreover, JMF RTP is implemented using UDP. Although this is a good solution for audio and video transmission (compared to a TCP-based one), its lack of reliability is inappropriate for user interactivity transmission, where every action must be sent to every participant and should definitely be received. Finally, JMF supports video rendering in “traditional” rectangular frames only. This forbids a smooth integration of naturally captured videos into the 3D virtual worlds. From the above, it is concluded that - JMF

[3] The number of implemented nodes is much bigger.

although it is very powerful - still misses some important characteristics to sufficiently support CVEs' implementation.
After an inspection of the missing features and characteristics we extended JMF capabilities in three ways to support:

- Transmission of MPEG-4 BIFS streams using JMF RTP,
- Reliable RTP multicasting and
- Video rendering on the surfaces of 3D objects

Fig. 2 shows the extended JMF RTP architecture, using a UML class diagram. Only the most important objects of the architecture are shown, to improve the readability of the figure. Only the sender part is shown. The receiver one has been designed similarly.

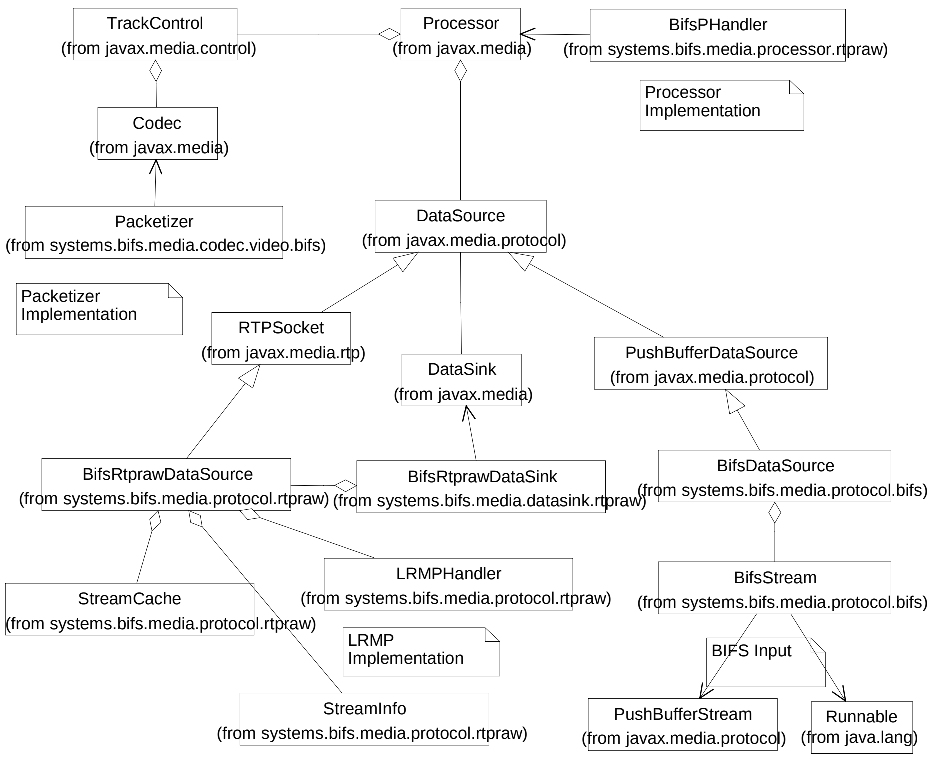
TrackControl
(from javax.media.control)
Processor
(from javax.media)
BifsPHandler
(from systems.bifs.media.processor.rtpraw)
Processor Implementation
Codec
(from javax.media)
Packetizer
(from systems.bifs.media.codec.video.bifs)
DataSource
(from javax.media.protocol)
Packetizer Implementation
RTPSocket
(from javax.media.rtp)
DataSink
(from javax.media)
PushBufferDataSource
(from javax.media.protocol)
BifsRtprawDataSource
(from systems.bifs.media.protocol.rtpraw)
BifsRtprawDataSink
(from systems.bifs.media.datasink.rtpraw)
BifsDataSource
(from systems.bifs.media.protocol.bifs)
LRMPHandler
(from systems.bifs.media.protocol.rtpraw)
BifsStream
(from systems.bifs.media.protocol.bifs)
StreamCache
(from systems.bifs.media.protocol.rtpraw)
LRMP Implementation
BIFS Input
StreamInfo
(from systems.bifs.media.protocol.rtpraw)
PushBufferStream
(from javax.media.protocol)
Runnable
(from java.lang)

Fig. 2. Extended JMF RTP architecture

JMF can be extended to transmit MPEG-4 BIFS streams. For this reason, it is necessary to add a new PlugIn in the JMF registry; a specific Packetizer to handle the packetization of MPEG-4 BIFS streams into RTP packets. The introduced Packetizer accepts the "bifs" Format as Input and the "bifs/rtp" (BIFS over RTP) as output. The Packetizer implements the Codec interface and is inserted in the Processor object that will handle this task (implemented by BifsPHandler, a Processor instance specialized for the BIFS streams). The Processor is created to handle the output of BifsDataSource. BifsDataSource, which extends PushBufferDataSource, encapsulates a BifsStream object. BifsStream is the object that provides the Processor with the data to be processed (packetized). Following the JMF model, the output of the processor can be obtained from an implementation class of the DataSink interface. BifsRtprawDataSink, which implements the aforementioned interface, encapsulates an extension of the RTPSocket, the BifsRtprawDataSource. This object handles the transmission of the BIFS with the RTP protocol. It encapsulates the LRMPHandler, which is a modified implementation of the LRMP protocol [10], to support reliable multicast delivery. LRMPHandler uses the RTCP Application packets to acknowledge the reception of the RTP packets (negative acknowledgement). StreamCache is a cache that stores the latest transmitted packages, in case a retransmission will be requested. StreamInfo stores the information, which is related to the status of each specific stream.

Fig. 3 shows the proposed solution for integrating video stream rendering on the surfaces of 3D objects. From the 3D modeling point of view, both the VRML and MPEG-4 BIFS specify that the MovieTexture node is responsible for defining the URL that points to the video that it is to be rendered on the surface of a geometry node. The MovieTexture node is a child of the Appearance node. From the implementation point of view, Java 3D provides an equivalent Appearance node to handle the appearance of a geometry node. The Appearance node encapsulates the Texture of the geometry node. In order to render video frames into the 3D object, the decoded video frame is processed from the VideoOn3DRenderer to transform it from the content of a Buffer object to the content of a BufferedImage, suitable to be rendered on the surface of the 3D object. This task is periodically repeated and gives the same feeling as a normal video display. It should be mentioned that VideoOn3DRenderer should replace the default renderer of the Processor, in the initialization phase of the MovieTexture node.

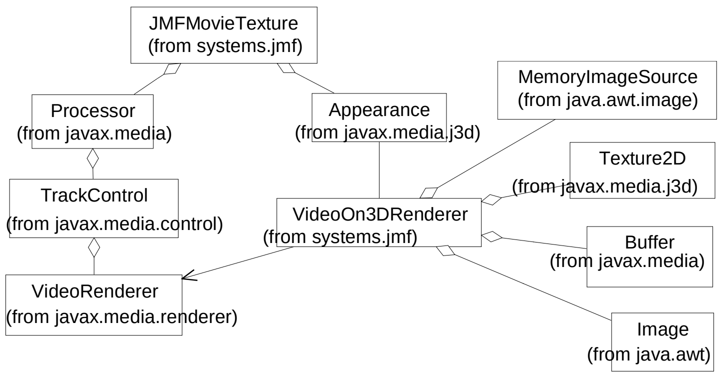

Fig. 3. Rendering video on 3D objects

4 A Case Study

The experiences described in this paper have been gained during the development of COSMOS [2]. COSMOS is an object-oriented framework that augments the rapid development of collaborative applications. It is developed using the aforementioned

software technology. In addition to the already described characteristics, COSMOS provides an implementation of the MPEG-4 DMIF (Delivery Multimedia Integration Framework) to handle the establishment and release of the communication sessions, for multicast scenarios. Also, an MPEG-4 Browser is an application that demonstrates COSMOS' capabilities. A screenshot of the browser is given in Fig. 4.
In this screen-shot, we demonstrate the rendering of a video stream in the surface of a 3D object. User collaboration is achieved by sharing the same content among different users. Their presence, as well as their interaction with the content of the shared world is denoted with avatars and their movements. DMIF handles the establishment or release of new streams while new users enter or leave from the shared world, dynamically.
COSMOS is capable of handling many users concurrently. Its decentralized architecture makes it a highly scalable system.

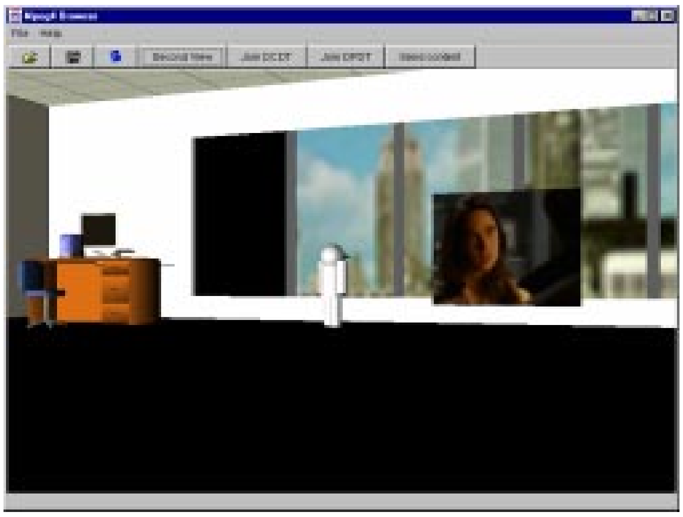
Fig. 4. MPEG-4 Browser screen-shot

5 Conclusions

CVEs' development is a very complicated task. International standards, like VRML and MPEG-4, define the specifications for providing interoperability among implementations. But still, implementation issues and hard to solve problems always arise in complicated multi-user multimedia systems. Wise choice of the implementation programming language can save a lot of work.
Java is a very powerful language that enables fast development of complicated systems. The language is consisted of a core part with the basic functionality and a set of extensions to handle special requirements. JMF is one of them and it provides a framework for multimedia processing, encoding, capturing, rendering and transmission. In addition, the Java 3D API is very helpful for the development of 3D applications. Finally, JavaCC can be used to develop text-based parsers.

The very careful design of the aforementioned Java tools enables their easy integration into systems. Extending those tools can become an efficient solution to rapidly develop prototype CVEs. COSMOS is an example prototype system, demonstrating the suitability of Java for developing collaborative virtual environments. It provides the proof of concept for the ideas introduced in this paper.

References:

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

[2] V. Darlagiannis, N. D. Georganas, "Virtual Collaboration and Media Sharing using COSMOS", 4th world multiconference on Circuits, Systems, Communications & Computers, July 2000

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

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

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

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

[7] ISO/IEC 14496-1 IS (MPEG-4), "Information Technology – Coding of audio-visual objects, Part 1: Systems", January 1999

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

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

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

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

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

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

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

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

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