INFORMATION TO USERS

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

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

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

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

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

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

Université d'Ottawa · University of Ottawa

Design and Implementation of an Advanced Telecollaboration System

By

Li Ding

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

Master of Computer Science

Under the auspices of the
Ottawa-Carleton Institute for Computer Science

University of Ottawa
Ottawa, Ontario, Canada
May, 2000

Abstract

In this thesis, we investigate two advanced features, which greatly enhance a telecollaboration system. The first one is real time audio and video synchronous communication among the participants. The second is the collaborative session recording service that enables the asynchronous collaboration. We describe our design and implementation of a real time audio and video tool JVC, and a recording tool JVCR.

JVC can capture, transmit and present the real time audio and video streams among all participants. The audio and video data are transmitted by RTP and IP multicast for efficiency. Different from other Internet-based audio and video tools, JVC is developed with Java and based on the Java Media Framework (JMF). Therefore, JVC is platform independent and well suited for the heterogeneous Internet environment.

JVCR is a WWW-based recording tool. It records all multimedia components of a collaborative session, such as user interactions with the JETS whiteboard developed by the Multimedia Communications Research Laboratory, University of Ottawa. It also records the associated live audio and video streams distributed by JVC in the collaboration session. A novel feature of JVCR is that it dynamically creates SMIL documents to specify the synchronization relationship of all captured components as well as the hypertext links to these components. Therefore, the recorded session can be easily accessed through the Web and played-back by any SMIL-enabled players. JVCR is also developed with pure Java for platform independence.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Nicolas D. Georganas. He not only provided me with direction and focus, but also gave me the opportunity to be his student. His continuous support, interesting discussions, and numerous encouragements made my research project most enjoyable.

Special thanks to Shervin Shirmohammadi. This project would not have been possible without his successful JETS. He also gave me valuable suggestions and comments for my project.

I would like to thank all members of Multimedia Communications Research Laboratory (MCRLab) for their cooperation and friendship. It is a real pleasure to work in MCRLab. I am proud to be a member of this winning team.

Finally, but by no means least, I would like to thank my parents, my wife Ms. Li Shi and my son Frank Ding for their unceasing moral support through the course of my studies.

Table of Content

Table of Figures

List of acronyms

AIFF: Audio Interchange File Format

API: Application Programming Interface

COR: Coordinator

CSCW: Computer Support Cooperative Work

GSM: Global System for Mobile

DBA: Database Access Wrapper

DG: Document Generator

HTML: HyperText Markup Language

JDK: Java Development Kit

JETS: Java-Enabled TeleCollaboration System

JMF: Java Media Framework

JPEG: Joint Photographic Experts Group

JVC: Java-Enabled videoconferencing system

JVCR: Java-Enabled Videoconferencing Recorder

JVM: Java Virtual Machine

MIDI: Music Instrument Digital Interface

MPEG: Moving Picture Experts Group

MBONE: Multicast Backbone

RF: Resource files

RS: Recorder

RTP: Real-Time Transport Protocol

RTCP: Real-Time Control Protocol

RTSP: Real-Time Stream Protocol

SD: SQL Database

SMIL: Synchronized Multimedia Integration Language

TTL: Time-To-Live

W3C: World Wide Web Consortium

XML: Extensible Markup Language

URI: Universal Resource Identifiers

URL: Uniform Resource Locator

VRML: Virtual Reality Modeling Language

Chapter 1 Introduction

1.1 Motivation

A telecollaboration system refers to a computer based system which supports coordination and cooperation of two or more people dispersed at different locations to perform a joint task. It emphasizes using the computer directly to facilitate human-to-human collaboration. Collaboration can be achieved synchronously or asynchronously. Synchronous collaboration is intended for simultaneous users who have real-time interactions, while asynchronous systems provide non-real-time communication, such as email and newsgroup.

A telecollaboration system has a long history. It belongs to the well-established research domain Computer Support Cooperative Work (CSCW)[26]. However, it did not achieve the expected success in the past. This is particularly true for widely-dispersed, cross-organizational working groups where problems of heterogeneity in computing hardware and software environments inhibit the deployment of such systems. This situation changes rapidly with the booming development of the Internet and Web in the 90s. Internet connects millions of computers distributed all over the world, introducing means for global computer communication. On the other hand, the Web provides a standard platform-independent interface for the user as well as for the backend servers. Moreover, Java, a platform-independent language, allows software to be delivered over the Web and executed on a variety of platforms without modification. Therefore, it is not surprising that a lot of Web based telecollaboration systems have been developed recently.

The Java Enabled Telecollaboration System (JETS)[1] developed at the Multimedia Communications Research Laboratory (MCRLab) is one of the first Web Telecollaboration Systems written in Java. JETS is a framework allowing collaborative work within any Java-enabled Web browser, by sharing Java applets synchronously. It can be used in many application domains, such as telelearning and

telemedicine. Currently, several shared Java applets or applications are developed based on JETS, such as a whiteboard and a VRML browser, just to name a few.

JETS successfully demonstrates the power of the Web as a platform of telecollaboration systems. However, it still has lots of room to be enhanced. First, JETS does not support live audio and video communication among all participants. Research has indicated that live audio and video play important roles in collaboration because a participant can get a " face-to-face" feeling and then greatly enhance the awareness of others [32]. Another limitation is that JETS lacks asynchronous collaboration support. A requirement for synchronous collaboration is that all participants should be present at the same time on their workstations. In some cases, this is too restrictive. Let's imagine that the participants may work in different time zones because their geographical distribution may be scattered all over the world. It is difficult for all participants to join the collaboration session at the same time. If the participant misses the session, he will miss all information about the collaboration because there is no session data recorded in the system.

Therefore, we believe that two advanced features will greatly enhance the collaboration productivity. One is to add real-time audio/video streams of participants to the synchronous collaboration space. Another is a recording service that can record all multimedia components of a synchronous collaboration session, including the interactions of shared applications as well as associated audio and video streams. Moreover, the recorded session can be instantly played-back. Combining synchronous and asynchronous collaboration tools, we can provide the participants a more effective collaboration environment.

1.2 Objective

We will research the integration of synchronous and asynchronous collaboration tools, together with real-time audio-video tools, using Java enabling technology for platform independence and portability. In addition, for asynchronous collaboration,

we will design a capture system for all multimedia interactions of participants. We will investigate:

- The use of one of the newest Java technologies, the Java Media Framework (JMF)[8], for capture and playback of real-time audio-video over the Internet.
- The use of real-time streaming protocols, standards and tools, such as the Real-time Transport Protocol (RTP) and the Synchronized Multimedia Integration Language (SMIL)[6] (a standard of the WWW Consortium);
- The dynamic creation of SMIL documents, composed of all multimedia components of a collaborative session. So we can capture a real-time collaborative session in SMIL and use SMIL-enabled players, for example, one of the more than 70,000,000 RealSystem G2 players [7] for later playback.

1.3 Contribution

The main contributions of this thesis are:

1. Design and implementation of the Java Videoconferencing Tool (JVC). JVC is a software tool to capture, transmit, and render the real-time RTP audio/video streams using multicast technology over the Internet. The most interesting feature of JVC is portability. Currently, most audio and video tools are platform dependent because they are written in C or C++. On the contrary, JVC is a pure Java tool. It can run over any platform that supports a Java Virtual Machine (JVM). Furthermore, it is easy to change JVC from the Java stand-alone application to a real time audio /video applet which can be downloaded from a Web server without installing it in the local machine. Therefore, JVC can be easily integrated with other Web based collaboration systems.

2. Design and implementation of the Java Enabled Videoconferencing Recorder (JVCR). JVCR is a recording system which can automatically capture all interactions of synchronized collaboration as well as associated audio and (or) video streams. Compared with the various recording tools currently available, JVCR has the following novel features:

- Open archive data format: the recorded sessions are stored as SMIL documents as well as multimedia components with well-known formats. The other recording tools store the session with ad-hoc file formats.
- Decoupling of recording and playback: the recorded sessions can be played-back by any SMIL player. On the contrary, the sessions recorded by other recording tools must be played-back by the recording tool or the original applications.
- Multiple protocols support: JVCR can record real time audio and video streams transmitted by RTP as well as JETS whiteboard events transmitted by TCP.
- Platform independence: JVCR is developed with pure Java and can run on any platforms supporting JVM.

1.4 Thesis Organization

The six remaining chapters are organized as follows: In Chapter 2, we introduce the technical background, which includes telecollaboration system architectures, the JETS architecture, the event broadcast mechanism, API and Whiteboard. We also briefly review IP multicast and the RTP protocol. Then we introduce the Java Media Framework. Finally, the SMIL specification language is discussed. The related work about the real time audio/video system over the Internet and the recording system of the collaborative system will be introduced in Chapter 3. We also point out the limitation of other works. In chapter 4, we present the design and implementation of JVC. The architecture, design and implementation issues will be discussed. The design and implementation of JVCR will be introduced in Chapter 5. In Chapter 6, we present the experimental results achieved with the JVC and JVCR. We also describe the problems faced during the testing phase. In Chapter 7, we summarize our work and suggest future work.

Chapter 2 Background

2.1 Telecollaboration System

2.1.1 Architecture

The architectures for collaborative system may be broadly categorized into two kinds of architecture: centralized and distributed. The centralized system is client-server based. The central server is used to pass information between clients as well as provide specific services. The advantages of the client-server based architecture are simpler clients, avoiding complex distributed algorithms, and ease of implementation. By contrast, in the distributed system, every site manages itself and broadcasts its message to other sites directly. A distributed system is more scaleable than a centralized system with the advantages of a potential increase in the maximum number of clients and avoiding an extra entity performing the duties of a server.

2.1.2 Collaboration Issues

The collaboration issues have been well recognized since the beginning of the 80's. Some key issues are briefly introduced in the following:

Floor control

A fundamental problem of a collaborative system is to coordinate the access to shared resources. Floor control realizes concurrency control for interactive, synchronous cooperation between people by using the metaphor of a floor. A floor denotes the temporary permission to access and manipulate resources, such as a shared drawing area or a video channel. In a session, floor control is mainly used to reduce non-determinism, setbacks, redundancy and inconsistencies, balance contributions among session participants in a fair manner, regulate the collaboration by a predictive and binding protocol and promote inter-group awareness, cohesiveness, and integrity [27]. The owner of a floor at a certain point of time is called the floor holder.

The floor policies describe how participants request for the floor and how it is assigned and released. Each system must decide the level of simultaneity to support, i.e., number of active users at once, and granularity at which to enforce access control. Four basic floor control policies are given in [29]:

- No control: each group member can access common resources without control.
- Implicit floor control: A floor is implicitly granted to a group member as soon as he or she starts to use a certain resource. The resource is then locked for other members. A certain time after the floor holder has stopped using the resource, the floor is automatically released.
- Explicit floor control: the floor is granted to a participant or is released from a participant only on his/her explicit request. The requests of other participants while the floor is locked have to be queued.
- Chair control: one person becomes chairperson or moderator of the collaborative group. He / She can grant or withdraw the floor for a certain resource at any time. The pending floor requests of participants have to be queued and monitored by the chairperson.

Session control

A session is a group of application instances currently working together in the collaborative mode. All applications belonging to the same session exchange information and share behavior. Session control administrates the multiple sessions with its participant and media, including initiation, pause, resume, and stop of sessions.

Message delivery

The collaboration applications interact with the system by message passing. If an application sends a message, this message will be delivered to all other compatible applications in this particular session. There are two major types of messages: control messages and application messages.

Control messages are generated by the system for communication between the server and control applications. These messages serve functions such as logging users into the system, establishing sessions, launching applications, etc. Control messages are invisible to user applications. Application messages are the main means of communication between user applications. The structure of application messages depends on the application, and is not interpreted by the communication system. The communication system is transparent to the application messages. The application - specific communication protocol is always defined by the application itself.

The message transmission approaches can be classified as unicast or multicast, which will be introduced in section 2.2.1.1.

2.1.3 Java Enabled Telecollaboration System (JETS)

JETS is a collaboration system designed for real-time sharing of Java applets. Using any Java-enabled Web browser, multiple users in a telecollaboration session are able to share generic applications in the form of Java applets.

2.1.3.1 JETS Functionality

JETS is a middleware framework to faciliate the development of a collaborative system. The main functionality provided by JETS consists of the following elements:

- floor control

JETS uses a centralized lock to coordinate the interaction with the shared application so that only one participant can interact with it at one time. When a participant wants to interact with an applet, the applet checks with the central server whether the shared application is currently available or not. The JETS server then checks the status of lock on this application. If it is available, then the applet locks the application so that other participants will be denied. After it is finished, it releases the lock so that other participants can use the application [2].

- message passing

The participant's interaction with the shared applications will be sent to the JETS server, and the JETS server then broadcasts it to all other participants. JETS also supports one-to-one communication by delivering the interaction of one participant to a specific participant. Each instance of the JETS shared application (except the one that fired the interaction) will recreate the received event represented by the message locally and achieve "What You See Is What I See" view of all shared applications.

JETS uses TCP/IP for reliable transmission. Because the message is short and the JETS server is a multithread system, the performance is very satisfactory [2].

- session management

There is a chairperson in each session. The chairperson has special privileges of controlling the application behavior and/or controlling access of other users to this session. JETS only permits predefined users to join the session. Each participant is assigned a certain level of privileges to access applications. During the session, a participant can request for higher access rights from the chairperson and receive those rights upon the chairperson's approval. The chairperson can also change the access rights of a participant dynamically during the session.

JETS is easy to use. To join a live collaboration session, the participants just need to go to a specific URL through the Web browser.

2.1.3.2 JETS Architecture

JETS is a typical client-server system. The application components with which users interact are distributed as applets, and a server application on the server manages shared state, synchronization, and dissemination of events. The centralized structure is a consequence of applet security restrictions, because an applet can only set up the communication with the host it is downloaded from. Furthermore, the centralized

architecture simplifies handling of many communication and synchronization issues, compared to distributed solutions.

A key feature of the JETS collaboration architecture is event multicast. The following figure [fig 2.1] shows how the event is multicasted among the identical client applets in JETS.

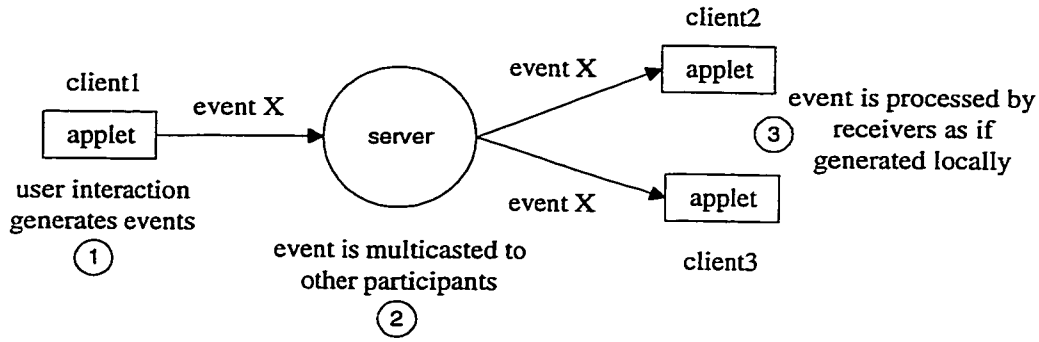


Figure 2.1 event multicast in JETS

When one user interacts with the applet, one or more events are fired. Each event is sent to the server, which multicasts it to other clients. The other clients then handle the event as if it were generated locally. By this way, they recreate the intended actions of the original client.

The event is packaged in the message with various length. To differentiate the semantics of the message, each message can be preceded by a "Token"[3]. For example, in following figure [fig 2.2] we see two types of message: one is a "draw" message; and the other is an "erase" message. Each of them has its own set of data to be transmitted. To differentiate between them, a token is attached to the beginning of each message.

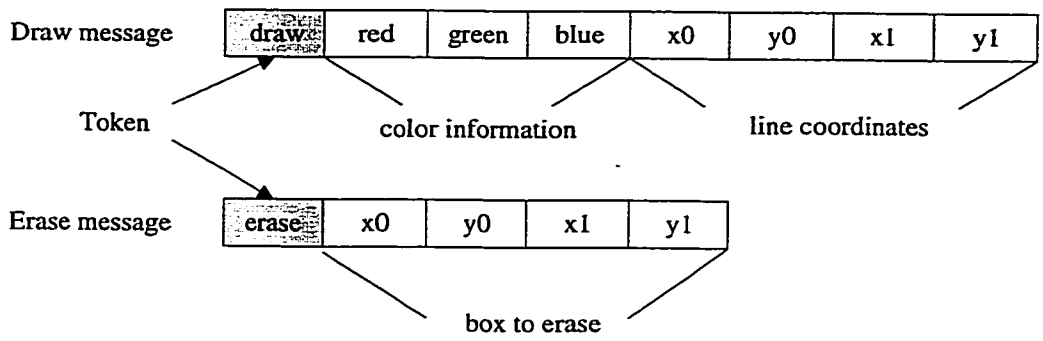


Figure 2.2 token used in the messages

JETS aims at providing a general framework, so it does not define a fixed set of tokens. On the contrary, an application developer defines the token according to the need of specific applications.

2.1.3.3 JETS application development

From the developer's viewpoint, JETS is a toolkit composed of a set of API used in the client side. These API are generic classes that provide collaboration abilities. The server side is hidden with the developer.

The JETS API currently provides three classes:

- AppletClient
- FrameClient
- ApplicationClient

AppletClient is the base class for a shared applet. FrameClient creates a Frame object that must be launched from an applet. It acts very similarly to AppletClient, except the applet which launches the frame must pass itself to the Frame through the Frame's constructor. JETS also supports Java shared applications. This kind of system should extend the ApplicationClient.

Each class provides very similar methods, which can be divided in four catagories:

- Event packet transmission
- Event handle
- Concurrency control
- Server connection

By extending the JETS API, we can develop the specific collaborative applications. Generally, the application developer is responsible for implementing event sharing by supplying routines that define the event structure, send the event and handle the received events from other participants.

Several collaborative applets have been developed based on the JETS API. One of them, Whiteboard, will be introduced in the following section.

2.1.3.4 JETS whiteboard

Whiteboard is a typical web collaboration application to provide a collaboration space for all participants. Multimedia objects such as JPEG or GIF images, H.263 video clips, etc., can be brought up from the Web server and presented inside the whiteboard. The participants can then annotate these images or paused H.263 video. They can also chat with each other in real-time through the embedded chat box of the whiteboard.

2.2 Real-time streaming media technology

Current Web and Internet technologies are limited when they come to support real time temporal media transmission and presentation.

For transmission, the Internet was not designed with real-time traffic in mind. Traditionally, the Internet uses TCP/IP. TCP provides reliable transmission between two hosts by using acknowledgements and retransmission. However, this mechanism

is not suitable for real-time audio/video transmission because packet loss causes retransmission that affects delay-sensitive data.

For presentation, the Web has integrated more and more types of media. Today, besides text and static images, we also have temporal media such as audio and video clips residing on Web pages. A critical issue is how to synchronize the presentation of different types of media. However, HTML, the current Web page authoring language, is designed for text and has no way of specifying the temporal relation of multimedia content. For example, using HTML, we can not express things like "five minutes into the presentation, show video X and keep it on the screen for ten seconds."

Therefore, two new components must be added:

- support for transmission of real time audio and video over the Internet.
- a format for authoring synchronized multimedia documents over the Web.

In the following sections, we briefly review the new techniques related to these two components.

2.2.1 Network technology

2.2.1.1 IP Multicast

IP Multicasting is the transmission of an IP datagram to a "host group"[40]. The host group is a set of one or more hosts identified by a logical group address which is a Class D address. Class D is indicated by "1110" at the start and its range is 224.0.0.0 to 239.255.255.255. The membership of a host group is dynamic. Hosts may join and leave groups at any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time. In addition, at the application level, a single group address may have multiple data streams on different port numbers, on different sockets, in one or more applications.

Multiple applications may share a single group address on a host. After joining a group, members receive all packets that are sent to the group address.

When a host wishes to join a multicast group, it issues an Internet Group Management Protocol (IGMP) [40] request to the respective multicast group address. IGMP is an extension of the IP protocol. It is used for the exchange of group membership between group members and multicast routers on a physical subnetwork. The multicast router for that subnet forwards that request to the other mrouters in the network so that such packets will get to this subnet and eventually be placed on the LAN where the host is connected. Multicast routers forward a packet to a network only if there are multicast receivers on that network.

IP Multicast is a more efficient way to use the network bandwidth than unicast. For instance, in one-to-many packet delivery, if we use unicast transmission, one host (the sender) sends the packet to only one specific single host (the receiver). Therefore, the sender must send separate copies to each receiver, and these copies all go through common network links. With IP multicast, the source sends only one copy, and the network ensures delivery to each member of the specified multicast group. Copies are made at just those points where paths diverge at routers.

Broadcast transmission is another approach in which one host sends the packet to all hosts on the network. However, it is a push method. The packet is sent to all nodes even if some nodes do not request it. Broadcasting does not consume the sender's resources any more than single unicasting does, but it inefficiently consumes network resources. On the contrary, IP Multicast is a pull method and the packet is sent to the hosts that request it.

2.2.1.2 MBONE

Multicast Backbone (MBONE) [4] is a virtual network on "top" of the Internet providing a multicasting facility to the Internet. MBONE was initially set up in 1993.

MBONE is composed of networks *(islands)* that support multicast. On each of these islands, there is a host running the *mrouted* multicast routing demon.

MBONE uses unicast "tunnels" to forward multicast packets among the islands of MBONE subnets through routers that do not support multicast. When sending the multicast packet through the tunnel, the multicast packet will be encapsulated inside the data part of a regular IP packet that is addressed to the mrouter on the other side of the tunnel. The receiving mrouter will strip off the encapsulation and forward the packet appropriately.

Multicast packet distribution across the Internet is controlled by specifying the Time-to-Live (TTL) of a packet. Each tunnel has a threshold, which is the minimum TTL that a multicast packet needs to be forwarded onto a given tunnel. The TTL field in a packet is decreased each time the packet passes through a mrouter. When TTL is lower than the threshold of the tunnel, the packet is dropped. The range of thresholds is between 0 and 255. The default value of TTL is 1 and the packet is multicasted inside the local subnetwork.

2.2.1.3 Real-time Transport Protocol (RTP)

RTP [5] is the Internet-standard protocol for the transport of real-time data. It has no dependencies on particular address formats and only requires that lower layers support framing and segmentation. RTP may run over either connection-oriented or connectionless lower-layer protocols.

RTP consists of a data and a control part. The data part of RTP is for real-time data transmission. It provides a standard format packet header which gives media specific timestamp data, as well as payload format information and sequence numbering, among other things. Each original RTP source is identified by a 32-bits SSRC (Synchronization Source Identifier) field. This field is assigned randomly by a sender. If a sender finds another sender who uses the same SSRC value in a session, both of them reallocate the field. Source related information is transferred with SD (Source Description) field.

The control part of RTP is called RTCP. Each RTP data flow is supplemented by RTCP packets. RTCP provides the following management functionality:

- QoS monitoring

RTCP packets contain the necessary information for QoS monitoring. Senders issue the SR (Sender Report) packet containing information useful for intermedia synchronization as well as cumulative counters for packets and bytes sent. This allows receivers to estimate the actual data rate. On the other hand, receivers must periodically send RR packets (Receiver Report) containing information related to the quality of each sender's transmission, such as the number of lost packets, fraction of lost packets and delay times.

- intermedia synchronization

The SR packets contain an indication of real time and a corresponding RTP timestamp. If the clocks of the different hosts are synchronized, these two values allow the synchronization of different media, such as audio and video.

- identification

RTCP carries a persistent textual identifier for an RTP source called the CNAME (canonical name). Receivers require CNAME to keep track of each participant and associate multiple data streams from a given participant in a set of related RTP sessions.

A RTP session is an association among the participants. It is defined by source and destination transport address pairs. Each transport pair includes a network address with port. For multicast UDP, RTP and RTCP use the same multicast address with two consecutive UDP ports, with the RTP port always being the lower, even - numbered one.

2.2.2 Synchronized Multimedia Integrated Language (SMIL)

2.2.2.1 Background

The Synchronized Multimedia Integrated Language (SMIL, pronounced as "smile") [6] is a recommendation for multimedia documents presentation over the Web released by W3C. SMIL 1.0 was accepted as a recommendation in June, 1998. Basically, SMIL specifies when and where to render the multimedia objects. The focus of SMIL is time. SMIL uses a timeline mode for synchronization. By using a single time line for all of the media on a page, their display can be properly time coordinated and synchronized. SMIL provides a rich collection of timing constructs and associated attributes to describe temporal relationships. Furthermore, SMIL also specifies how a SMIL document can be customized according to different rendering system abilities and settings.

The SMIL document is XML [10] compliant. The document has a tree structure, which is very similar to that of HTML. The nodes give the temporal and other compositional structure of the presentation. The leaves represent the media objects. Similar with an HTML document, a SMIL document is a text file. It can be created with any text editor manually or created with a program on the fly.

A SMIL document is played-back by a SMIL player which can be standalone application or a 'plug-in' of Web browser. Several SMIL players have been implemented in research communities and the marketplace, such as RealNetwork's RealPlayer[7] .

2.2.2.2 SMIL Document structure

A SMIL document begins with the *smil* element, which serves as the parent of all other elements. The document consists of two parts: Head and Body. The head part specifies meta information and all available regions. The body part defines how to present each media object. The document structure can be shown as follows:

```
<smil>
   <head>
        specify meta information and spatial layout of rendering surface
   </head>
    <body>
        specify the temporal and spatial relationship of media objects
    </body>
</smil>
```

The basic specification unit of SMIL document is the element. All elements are labeled with a tag. Each element has the associated attributes. The basic structure of an element is:

```
<element-name attributes = "value" >
        other elements
< /element-name>
```

An element can be nested inside the appropriate high-level elements:

```
<element-name attributes = "value" >
< element-name attributes = "value" />
< element-name attributes = "value" />
< /element-name>
```

In this case, the nested element is called the child and the high-level element is called the parent.

The following figure [fig2.3] shows the elements defined in SMIL.

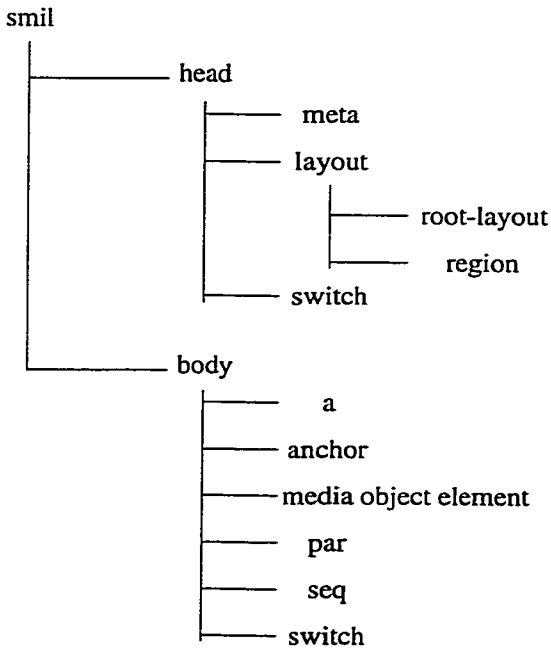


Figure 2.3 the SMIL elements

Document head

The head part consists of *head*, *layout*, *root-layout*, *region*, and *meta* elements. The *head* element contains information that is not related to the temporal behavior of the presentation. The *layout* element determines how the elements in the document's body are positioned on an abstract rendering surface. The rendering surface can be either visual or acoustic. The *root-layout* element determines the value of the layout properties of the root element, which in turn determines the size of the window in which the SMIL presentation is rendered. The *region* element controls the position, size and scaling of media object elements. SMIL defines the screen area as several rectangular regions. Several regions can overlap. Each region has an ID, which can be referred to by media object to specify where it will be presented. Regions are static and can not be redefined dynamically. The *meta* element defines the meta information of the document, which is specified as property/value pairs, such as author, expiration date, a list of key words, etc.

Document body

The *body* element contains information that is related to the temporal and linking behavior of the document.

SMIL can include many types of media objects in the document. Each type of media objects is defined as a media object element, such as *ref*, *animation*, *audio*, *img*, *video*,*text* and *textstream* element. Media objects are specified by their Uniform Resource Identifiers (URI)[17].

To specify the temporal synchronization among the media objects, SMIL defines a *par* element and a *seq* element. The media object elements nested in a *par* element can overlap in time. On the contrary, the media object elements nested in a *seq* element are presented sequentially. By nesting *par* and *seq* elements, we can specify the complex synchronization relationship.

SMIL uses the timeline model to specify the begin, duration, and end time of each media object. For a continuous media object, such as audio and video, the *clip-begin* attribute specifies the beginning of a sub-clip of a continuous media object as offset from the start of the media object. The *clip-end* attribute specifies the end of a sub-clip of a continuous media object. The time value can be either delay value or element-event value.

A delay value is a clock-value measuring presentation time. The semantics of a delay value depend on the element's first ancestor, which is a synchronization element. If this ancestor is a *par* element, the value defines a delay from the effective begin of that element. For example, the semantics of the following fragment can be shown by figure 2.4.

```
<par>
    <video id="a" begin="10s" src="video" />
</par>
```

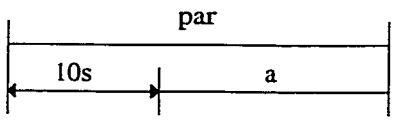


Figure 2.4 Using a delay value within a par element

If the ancestor is a *seq* element, the value defines a delay from the effective end of the first predecessor. Let's say we have the following SMIL fragment:

```
<seq>
   <video id="video1" src="video1" />
   <video id="video2" begin="4s" src="video2" />
</seq>
```

Figure 2.5 shows the semantics:

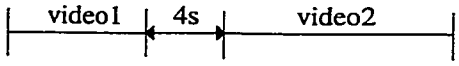


Figure 2.5 using a delay value within a "seq" element

The event-value is used to specify that an element begins when a certain event occurs.

```
<par>
  <audio id="a" begin="10s" ... />
  <img  begin="id(a)(5s)" ... />
</par>
```

The following figure shows its semantics:

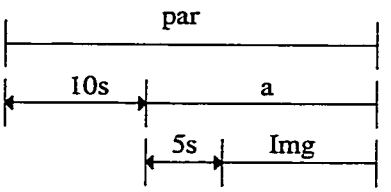


Figure 2.6 Synchronization attribute with element event value

Due to its integrating nature, SMIL also supports hyperlinks to other SMIL or HTML documents through *a* and *anchor* elements. The functionality of the *a* element is to associate a link with a complete media object. The *anchor* element is similar to *a* element, but allows associating a link destination to spatial and temporal subparts of a media object.

A SMIL document provides adaptation ability by specifying the *switch* element. The *switch* element specifies a set of alternative children elements. Each synchronization child element specifies the attributes for system capabilities and settings. These attributes describe dynamic aspects of the environment which can then be tested at run-time, such as approximate bandwidth, captions, presentation language, rendering screen size, etc. When presented, the SMIL player checks the corresponding system attributes and selects only one acceptable element to render. For example,

```
...
<switch>
  <audio src="audio-better-quality" system-bitrate="16000" />
  <audio src="audio-average-quality" system-bitrate="8000" />
</switch>
...
```

This fragment chooses between audio resources with different bitrates.

2.3 Java Media Framework (JMF)

The JMF is a standard extension of core JAVA technology to enable the display and capture of multimedia data within Java applications and applets. It specifies a unified architecture, messaging protocol and programming interface for playback, capture and conferencing of compressed streaming and stored timed-media including audio, video, and MIDI across all Java-enabled platforms. It also provides the ability to access and manipulate media data before it is rendered.

2.3.1 Architecture

The design goal of JMF is to provide an easy-to-use API for incorporating time-based media into Java programs while maintaining the flexibility and extensibility required to support advanced media applications and future media technologies. The following figure shows the high-level architecture of JMF.

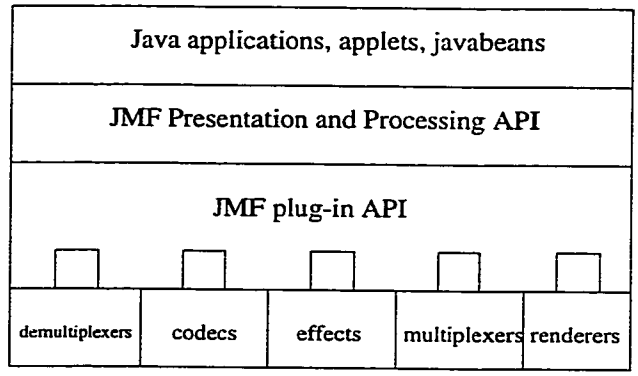


Figure 2.7 the high-level architecture of JMF

As we can see, JMF consists of two layers of API. A higher-level API, called the JMF Presentation and Processing API, manages the capture, presentation, and processing of time-based media. The lower-level API supports the seamless integration of custom processing components and extensions. JMF can be integrated with Java application, applets or javabeans seamlessly.

2.3.2 Main classes

The main classes and interfaces of high-level API are introduced briefly as following:

MediaLocator: describes the location of a media stream. A MediaLocator is closely related to an URL, but identifies stream parameters.

Datasource: encapsulates the location of media, the protocol and software used to deliver the media.

Player: processes a stream of data, reads data from a media source and renders it at a precise point in time.

Processor: a special type of player that can provide control over how the media data are processed before they are presented.

Datasink: reads media content delivered from a *DataSource*, and renders the media to some destination, such as a file.

Manager: a factory class which constructs the instance of *Player*, *Processor*, *DataSource*, and *DataSink* class.

The low-level API consists of five "plug-in" components.

Demultiplexers: extracts individual tracks of media from a multiplexed media stream.

Multiplexers: joins individual tracks into a single media stream.

Codec: performs media stream encoding and decoding.

Effect filters: modifies the track data in some way, often creates some special effects. Depending on when they are applied in relation to the codec plug-in, effect filters can be classified as post-processing or pre-processing effect filters.

Renderers: delivers the media stream to the presentation device. Typically, the video presentation device is the computer screen and the audio presentation device is the speaker.

To support real-time transmission, JMF includes the RTP APIs as the integrated part. The first version of the JMF RTP APIs support receiving and presenting RTP streams. In the current JMF 2.0 version, the RTP APIs also support the RTP stream transmission over network and store them in files. Most standard file formats and media types can be supported, such as AIFF, AU, AVI, GSM, MIDI, MPEG, RMF, WAV, QuickTime. RTP APIs also support low-bit-rate codec, such as H.263.

The main classes and interfaces of RTP API are:

RTP stream: an interface that represents a series of data packets originated from a single host. There are two sub-interfaces of RTPStream: *RecvStream* and *SendStream*. The first one represents a stream that is being received from a remote participant. The second represents a stream being sent by a local participant.

RTP participant: represents one participant in an RTP session. A participant that sends no data is called a Passive Participant. Otherwise, it is called an Active Participant.

RTCP Report: represents both RTCP SR and RR packets. SR report is sent by an active participant and RR report is sent by a passive participant.

SessionManager: coordinates an RTP session by keeping track of the session participants and the streams that are being transmitted. It defines methods that enable an application to initialize and start participating in a session, remove individual streams, and close the entire session. The session manager also handles the RTCP control packets.

2.3.3 Event model

JMF uses a structured event reporting mechanism to keep JMF-based programs informed of the current state of the media system and enable JMF-based programs to respond to media-driven error conditions, such as "out of data" and "resource unavailable". Whenever a JMF object needs to report on the current conditions, it posts a *MediaEvent* which is subclassed to identify many particular types of events. To receive notification, the JMF-based application implements the appropriate

listener interface and registers itself as the listener of the object which posts that event.

To facilitate the processing of RTP stream, the JMF defines several RTP-specific events and listener interfaces:

ReceiveStreamListener: Receives notification of changes in the state of an RTP stream that's being received. It can listen to the following events:

- NewReceiveStreamEvent: new receive streams are detected.
- ActiveReceiveStreamEvent: the transfer of data has started.
- InactiveReceiveStreamEvent: the transfer of data has stopped.
- TimeoutEvent: the data transfer has timed out.
- RemotePayloadChangeEvent: the format or payload of the received stream has changed.
- StreamMappedEvent: a previously orphaned receive stream has been associated with a paricipant.
- ApplicationEvent: an RTCP APP packet has been received.

SessionListener: Receives notification of changes in the state of the session. It listens to the following events:

- NewParticipantEvent: a new participant has joined the session.
- LocalCollisionEvent: the participant's synchronization source is already in use.

SendStreamListener: Receives notification of changes in the state of an RTP stream that's being transmitted. It listens to the following events:

- NewSendStreamEvent: a new send stream has been created by the local participant.

- ActiveSendStreamEvent: the transfer of data from the DataSource used to create the send stream has started.
- InactiveSendStreamEvent: the transfer of data from the DataSource used to create the send stream has stopped.
- LocalPayloadChangeEvent: the stream's format or payload has changed.
- StreamClosedEvent: a stream has been closed.

RemoteListener: Receives notification of events or RTP control messages, which are received from a remote participant. It listens to the following events:

- ReceiverReportEvent: an RTP receiver report has been received.
- SenderReportEvent: an RTP sender report has been received.
- RemoteCollisionEvent: two remote participants are using the same synchronization source ID (SSRC).

2.3.4 Scenarios

JMF provides an easy-to-use framework to handle the stream media. The main design task of a JMF-based application is to implement the appropriate interface and callback functions. In the following section, we introduce how to use JMF to process real-time media stream.

2.3.4.1 Capturing

JMF supports live video capture from camera and audio capture from microphone. The capture device is represented by a *CaptureDeviceInfo* object. It can be accessed through the *CaptureDeviceManager*, which is the central registry for all of the capture devices available to JMF. To capture media stream from a particular device, the first step is to get the device's *MediaLocator* from its *CaptureDeviceInfo* object. Then we can use this *MediaLocator* to construct a *Player* or *Processor* directly, or

construct a *DataSource* that can be used as the input to a *Player* or *Processor*. Then we can initiate the capture process by starting the *Player* or *Processor*.

The *Player* only renders the captured media stream. The *Processor* has more functionality: it can explicitly process or store the captured media stream.

2.3.4.2 Transmitting

JMF provides two classes to manage the RTP stream transmission: *DataSink* and *SessionManager*. The *DataSink* can only transmit the first stream in the *DataSource*. If we want to transmit multiple RTP streams in a session or need to monitor session statistics, we need to use the *SessionManager* directly. Because the later approach provides more functionalities, we introduce it as the following.

To transmit the RTP stream, the first step is to create a *Processor* and set the track format of processor to a RTP-encoded format, such as RTP-H263. Then we construct a *SessionManager* with the session's multicast address, audio, and video ports. After *SessionManager* is initialized, a *SendStream* can be created for the specified *SourceStream* by calling *createSendStream* on this *SessionManager*. Finally, we control the transmission through the *SendStream* object, which provides the methods to start, stop, pause and resume the transmission of the specific stream.

2.3.4.3 Receiving

In order to receive the RTP streams, we create a *RTPSessionManager* to join the multicast session. *RTPSessionManager* automatically detects any newly received streams from remote participants. Then it will post a *ReceiveStream* event. The application which registers as a listener of this event will be notified and then fetchs the stream from this event.

2.3.4.4 Presenting

Each stream is presented by a *Player* object. The *Players* are created from the received RTP streams by retrieving the *DataSource* from the received stream and passing it to the factory class *Manager*.

To present the stream, we should get the visual component and control component of the stream by calling *getVisualComponent* and *getControlComponent* method respectively . The visual component is the visual representation of the stream and can be displayed in a Java AWT container. The control component allows us to control the media presentation. Furthermore, we can get the control panel, the GUI of the corresponding control component, through *getControlPanelComponent* method. For example, a *Player* might be associated with a set of buttons to start, stop and pause the media stream, and a slider control to adjust the volume. If we prefer to define a custom user-interface, we can implement custom GUI components and call the appropriate *Player* methods in response to user actions.

2.3.4.5 Exporting

To export the received RTP streams to a file, we first retrieve the *DataSource* from the *ReceiveStream* and use it to create a file writing *DataSink*. If we want to transcode the data from the format used in transmission, we can do a "filtering" processing. We can use the *DataSource* to construct a *Processor*. Then we set the track formats of the *Processor* to perform the desired encoding, get the *DataSource* from this *Processor*, and construct a *DataSink* with the *DataSource*.

Chapter 3 Related works

In this chapter, we will review the related works with this thesis. In section 1, we introduce existing real-time audio and video systems over the Internet. The recording application of the collaboration system will be presented in section 2. We also compare our work with the related works in each section.

3.1 Real-time audio and video applications over the Internet

Currently, the most pervasive real-time audio and video applications over the Internet are MBONE tools, which are fostered by the deployment of IP Multicast. The key features of these tools are the usage of Application Layer Framing (ALF) [33] and the common support of the RTP. These applications are focused on scalability of very large scale conferencing over the Internet, and use an approach referred as loose control of the conferencing in which senders may not know who is receiving. If there is any floor control, it can only be through rough consensus, rather than by deterministic algorithm. MBONE tools don't require explicit conference control mechanism. Typically, the only conference control information needed is the distributed RTCP session information, such as an approximate membership and reception quality reporting.

The first publicly available MBONE audio tools were VAT[34] and VT. These tools provide speech communication using transmission speeds up to 64kb/s. The audio quality provided is degraded by network packet loss because of the best-effort Internet service. RAT[16][35] was designed to address this problem. Available video tools include IVS, NV and VIC[15] which provide slow-scan video at a quarter of TV resolution. The frame rate achieved by these tools is limited by the processing power of the transmitting workstation and the available bandwidth in the network. VIC, which is the most popular of the three, produces around 300kb/s out of an image of a moving person, using a simplified H.261 software codec. Each of the media tools in an MBONE session operates its own RTP session. To provide co-

ordinated control of the different tools, a local conference bus is used within each end-host. The individual tools can exchange information and control messages over the bus.

Other application families for real-time audio and video applications are based on ITU H.323 standards. An example is Microsoft's NetMeeting[41]. These tools are focused on tightly coupled conferences with few members when each participant knows where the other participants are at all times. All participants should have complete ordered and reliable floor control.

However, there are few real-time audio and video tools written in Java, although Java has been recognized as a language to develop heterogeneous network-centric systems. Most of them are written in C or C++. Some tools provide a Java-based user interface but still hook up to native C or C++ code through the Java Native Interface. Although C and C++ based systems may have better performance than that of Java, the drawback of them is the dependence of platform which has been recognized as a main problem for collaborative system [26].

Different from the above systems, our JVC is based on Java Media Framework, an extension of core Java technology. JVC can be integrated with other collaboration tools seamlessly. JVC provides full functionality to capture, present, and store the real-time audio and video streams transmitted by RTP over IP multicast. To facilitate collaboration through different hardware and software environments, JVC also supports a lot of audio/video codecs.

3.2 Recording application

MBONE VCR [19] is a recorder application for MBONE videoconferencing systems. It provides functions to record and playback MBONE sessions with

multiple multicast multimedia data streams from different MBONE collaboration tools at various locations. Examples of these tools include RAT, VIC and WB. Whenever it receives a RTP packet, it dumps the packet to local disk file. During recording, the MBONE VCR will synchronize audio and video streams based on information provided by RTP. To play back recorded sessions, MBONE VCR loads the dumped packets from the file, sends these packets to the original applications and recovers the original timing of all the media streams included in a session. The MBONE VCR uses a time-line model to synchronize all data streams. It builds a time scale from 0 to the length of the session, mapping the various time-stamp information into one particular format to allow random access. However, the MBONE VCR is a stand-alone signal-user application and cannot be accessed from remote sites. MBONE VCRoD [20] is the successor of MBONE VCR. It preserves the basic function of MBONE VCR but uses flexible client-server architecture. Therefore, the user can remotely control the recording and replaying from the client machine.

MASH[31] also has an archive system for MASH MBONE collaboration tools. The archive system is composed of a stand-alone recorder and player, and a set of distributed archive servers to record and play back RTP data (MASH audio and video) and SRM[36] data (MASH whiteboard). The recorder and player are intended for locally controlled playback and recording. The recorder accepts an SDP[37] announcement or a set of multicast addresses to specify which session should be recorded. The recorded sessions are stored in three types of files :

- data files: stores all data packets of one media type from a single source.
- index files: contains a mapping from time to data file position, and allows fast random access. Index files are generated for each data file.
- catalog file: lists all of the component data and index files for an archived collaboration session.

The player accepts a catalog file or the specification of a number of individual data files. The user also specifies a set of multicast addresses for playback. Standard playback functionality of random access and pause is also implemented.

There are some ongoing research projects about MBONE recording. For example, [14] presents a distributed recorder system for MBONE videoconferencing recording.

Although the above applications have some differences in functionality, a common limitation of these applications is that they are designed only to record the multicast streams. This is not enough for the recording of our collaboration system composed of JETS and JVC. As we mentioned before, JETS is a tightly-coupled tool using TCP/IP as the underlying protocol and all events are broadcast by a central server. On the other hand, JVC uses RTP over IP multicast to multicast the audio and video data. Another limitation of the above applications is their ad hoc archive data format that will cause the lack of interoperation of recorded session between different collaboration systems. For example, the MBONE VCR can not play back the MASH recorded session and vice verse.

Besides the MBONE recording applications, some recording applications for Non-MBONE collaboration system can be found in the literature.

[13] and [30] introduce a client-server based recording system to record and replay the session of CORONA [23] collaborative system. The recording system includes a session manager and multiple stream controllers. Each stream controller captures and dispatches events to its application. The session manager coordinates the services of the various stream controllers. The applications should implement a set of predefined interfaces in order to be recorded.

A session is captured in so called session object. The session object contains information about user interaction with the application, audio annotations, application states and resource reference. The recorder provides a logical time system to support time-stamping of events. A session object is stored as a directory that is composed of a set of files:

- a session header file containing the metadata of a session;
- a measurement file containing the data needed to support synchronized replay of the session;
- a resource directory;
- a header file and a data file for each stream.

Besides ad-hoc archive data format, the major limitation of their system is that the recorded applications should be recording-aware.

TANGO [24] is a Web-based collaborative system that supports both synchronous and asynchronous collaboration modes. Session record and playback capability has been designed into the system as its fundamental component. This capability applies to both events and data streams and can be used to review collaborative sessions in asynchronous fashion. Playback / review capability is also provided for real-time continuous data streams such as audio and video. The recording is implemented by an event log method. In TANGO, all control and application messages must go through the main server, so all of them with the date, exact time, and sender information can be captured and recorded in the database. When replaying the recorded session, the database acts as a source of information. Since it keeps applications message as well as control messages, user status, session's configuration and applications running can be restored. However, as the other recording tools, TANGO uses an ad-hoc data format to record the session. Only TANGO itself can play back the recorded session.

[11] presents a Web-based conference Minute System for group collaboration. This system can record the shared applications as well as real-time Audio/Video systems based on their collaboration framework. [12]

The system uses two recorders. One is a Java-written recorder for recording shared applications. Another one, called AV recorder, is a C-written recorder for recording and replaying the audio/video data of the videoconferencing.

The recorded session is modeled as a structured progress flow that consists of a set of nodes. Each node is linked with a minute object, which includes corresponding input events and audio/video data. A scribe is responsible for identifying the event manually and maintaining the progress flow. Events begin with the keywords that are spoken by the session owner with predefined semantics. Whenever the scribe hears a keyword, he / she chooses the corresponding type of node in a keyword list to add a new node at the correct location in the flow.

When the conference ends, the captured minute objects are converted into HTML format, which can be accessed from a Web browser. A Java applet invokes the "native" C-written conference minute system to show the progress flow. The reviewer clicks the node in the flow to replay the recorded input events and audio/video data. The recorded timestamps are used for achieving synchronization between the Java player and AV recorder.

This system presents a way to record the session and store it with Web available format. Although it is somewhat similar with our work, there are some significant differences. First, the MINUTE system is suitable for the collaboration which has a specific agenda and is well organized. However, it is not suitable for other kind collaboration. Secondly, the system needs a lot of user intervention to manually recognize the collaboration events and put them in the right place in the flow chart. Different from MINUTE, our JVCR is easy to use. The scribe only needs to click the button to start the recording. Then it automatically records the specific data streams. Furthermore, MINUTE is not an open recording system. Although an HTML document is used to store the minute object, it only specifies the links to the recorded session. To play back the recorded session, the MINUTE system sends the archive data back to the original applications that will play them back. By contrast, JVCR uses SMIL documents to store the recorded session, which specify not only the links to specific recorded multimedia components, but also the synchronization relationship among them. The multimedia components themselves are recorded by

JVCR in well-known formats. Therefore, the recorded sessions can be played back by all SMIL-enabled players. Finally, MINUTE does not support the recording of an RTP stream. JVCR, as we mentioned before, can record the streams transmitted by RTP and TCP.

[21][22] also present the recording systems to capture and play back the live collaborative activities, usually in a meeting room with some special equipment. However, JVCR is used to record the desktop collaborative applications.

Chapter 4 Design and Implementation of JVC

In this chapter, we will present the design and implementation of our Java-Enabled Videoconferencing Tool (JVC). We first introduce some key design issues of JVC in section 4.1, then we present the architecture and components of JVC in section 4.2. JVC implementation will be presented in section 4.3 . Finally, we introduce the operation of JVC in section 4.4.

4.1 General specification

4.1.1 Goals

JVC provides desktop videoconferencing abilities. The design goals can be summarized as the following.

- *Efficient*

Streaming over the Internet needs very high bandwidth. JVC should be efficient in order to provide acceptable performance under current Internet connection.

- *Flexible*

JVC should be a general tool which can be easily integrated with other Web telecollaboration systems, such as JETS.

- *Platform independent*

JVC is designed as a Web videoconferencing tool. As we know, the Internet is a highly heterogeneous computing platform. To be accessible through the Internet, JVC should be platform independent.

JVC should provide the following functionality to participants:

- Join the real-time audio and (or) video session.
- Leave the session.

- Capture and send the live audio and (or) video streams to other participants. The streams can be paused and resumed under participant control.
- Receive the audio and (or) video streams of other participants.
- Play the received audio and (or) video streams which can be paused and resumed under the participant control.

4.2 Design decisions

JVC is designed based on JMF. As a framework, JMF provides a set of easy-to-use APIs which make multimedia processing much easier than developing it from scratch. More importantly, JMF defines the general software architecture to process and control audio, video and other time-based data within Java applications and applets. As a JMF-based software, we design the capture and playback of live audio and video within the framework provided by JMF, which basically is to implement the appropriate interfaces and callback methods as well as process the media data according to the JMF specification.

However, JMF does not specify application related funcationality. The following issues are important to JVC design.

4.2.1 Real-time audio video transmission

To address the challenge of transmitting real time audio and video among all participants, we make use of RTP over MBONE. As we mentioned in Chapter 2, this is a flexible and highly efficient transmission approach to deliver continuous media over the Internet.

The issue here is whether we transmit audio and video streams separately or interleave them and transmit them as one stream. Because media streams have different creation, transmission, and playback characteristics, they have different QoS requirements. In order to control audio and video separately, JVC transmits audio and video as two separated streams to the same multicast address with different port numbers. The inter-stream synchronization of audio and video streams can be achieved by timestamps in RTP data packets and RTCP control packets.

Given that video and audio of a participant are delivered as two separate streams, we need a way to identify that they are coming from the same person. In RTP, SSRC field is used to identify the stream source. SSRC is generated randomly. However, it may be changed if the SSRC is already used by another stream. Therefore, we don't use SSRC but use the CNAME field of the RTP packet to identify the source of a stream. To assign a unique CNAME for every participant, we construct CNAME as: user_name @ host_address, in which user_name is the user login name of the collaboration session. host_address is the IP address of the endpoint.

4.2.2 Control structure

There are two choices for conferencing control structure: centralized or distributed. In the centralized system, there is a central component for membership management, chair control and floor control. In the distributed system, all sites are equal. Any site can send if it wishes, and sites causing trouble are simply muted at the receiver side[9].

The trade off between the centralized system and the distributed system is control-ability and scalability. Because a collaboration system over the Internet may be very large eventually, scalability of the system is important. Therefore, totally-distributed architecture is more flexible. However, there are a number of occasions when a more cooperative(and possibly restrictive) conference control scheme would be preferred, for example, to trade-off limited video bandwidth between a speaker and questioners under the control of a conference chair. JVC uses a loosely-coupled structure composed of clients and a central server.

Most conferencing systems with a central server have a "thin-client, fat-server" structure. The server does all of the management work and the client usually implements a graphics interface. Although it is easy to control the whole system, the server is the potential bottleneck. An important feature of our structure is "fat-client, thin-server". The conferencing management functionality is divided into two parts

between the client and server. The server is only in charge of admission control to check who is permitted to join the session and access the sending or receiving media services. However, the server does not maintain the dynamic participants list, which will be distributed by the RTCP packets instead. The client in each site has two layers: application layer and Audio/Video service layer. The Audio/Video service layer is lightweight: it sends RTP streams to the multicast address or receives the streams from it. It also listens to the RTCP packet to get membership information of other participants of Audio/Video session. The Audio/Video applications are built above the Audio/Video service layer and connect with the JVC server. The JVC clients communicate with the JVC server through TCP/IP to ensure error-free transmission.

4.3 Design of JVC

4.3.1 Architecture

According to our goals and design decisions, we present our proposed architecture of JVC, as shown in the following figure [fig 4.1].

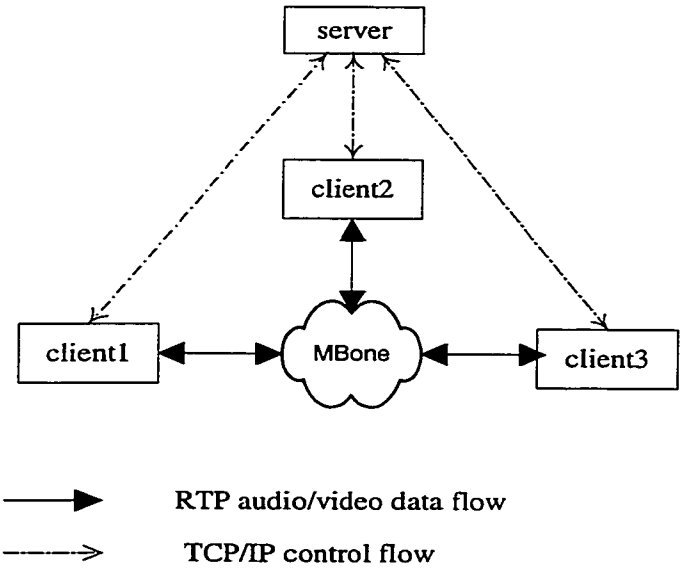


Figure 4.1 the JVC architecture

The JVC is a multi-user client/server system. The JVC server is in charge of admission control. The client is in charge of media transmission. The JVC server and

clients have bi-directional communication. The server receives the service access request from clients and returns the admission result to them. JVC uses different protocols to transmit control data and media data. The control data are transmitted through TCP/IP for reliable transmission. The media data are transmitted through RTP over MBONE. The media data are distributed among all clients by multicast without involving the server.

4.3.2 Design of the JVC server

The JVC server is similar to the JETS server [2]. However, there are two important differences. First, the JVC server only has a signal channel [2]. Because the audio/video data are distributed using multicast, the JVC server does not need a data channel that is used in the JETS server to collect the data from one client and distribute it to other clients. Another difference is the floor control mechanism. The JETS server uses a semaphore to enforce consistency of shared resources. The semaphore works in "one at one time" style, which means only one client can access the shared resource at any given time. By contrast, the objective of floor control in the JVC server is to control the workload of the system by granting or refusing permission to a client to transmit live media data into the system. The JVC server defines the maximum number of *tokens* which is an abstraction of how many audio and video channels are supported to transmit streams concurrently by the JVC system. The JVC server can grant *tokens* to multiple clients, if the number of issued *token* does not exceed the maximum number.

The JVC server is designed as a multithread application to maximize the real time behavior. It is composed of the *JetsStreamServer*, *subStreamServer* and *Monitor*.

JetsStreamServer

The *JetsStreamServer* works as a thread dispatcher. It creates a ServerSocket with specific port and listens for a connection to be made to this ServerSocket and accepts

it. When a client connects to the ServerSocket, *JetsStreamServer* returns a Socket object and gets the DataInputStream and DataOutputStream objects of the connection with the specific client through the Socket. Then it spawns a new instance of *subStreamServer* to serve this client.

subStreamServer

The *subStreamServer* gets the *DataInputStream* and *DataOutputStream* objects from the *JetsStreamServer*. It then receives the client's request through *DataInputStream*. If the request is to apply for a token to send A/V data, it uses admission algorithm to decide whether to grant or refuse the request. Currently, JVC uses a simple algorithm: "first come; first served". If the permission is granted, then the number of free tokens decreases by one. Otherwise, the request is refused. The result of the request is sent to the client through *DataOutputStream* object.

Monitor

The *Monitor* keeps track of the number of available *tokens*. A *token* is a shared resource and the number of available *tokens* may be updated concurrently by several instances of *subStreamServer*. Therefore, the *monitor* provides a set of synchronized methods to access the *token*.

4.3.3 Design of the JVC client

The JVC client is based on JMF API. It has five key components: the *Manager*, the *Sender*, the *Receiver*, the *Renderer* and the *Control panel*.

The architecture of JVC client is shown in the figure 4.2.

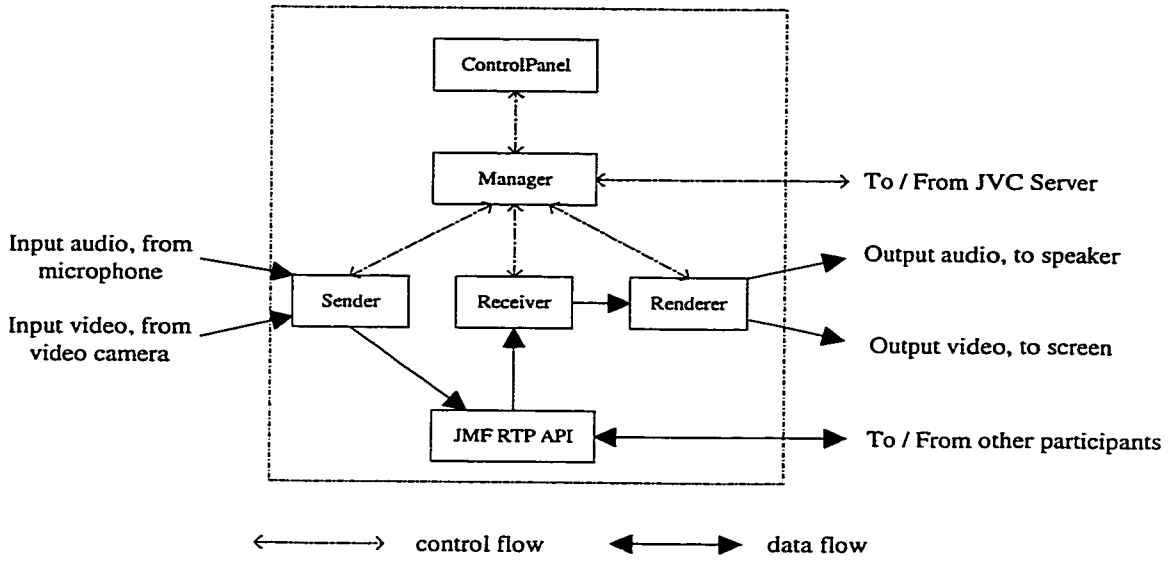


Figure 4.2 the architecture of the JVC client

- *Manager*

The *Manager* coordinates the behavior of all other components. It launches the *Sender* and the *Receiver* on demand as well as parses and dispatches the user control command to the *Sender* or the *Receiver*. The *Manager* also handles the control data communication with the JVC server. It creates a *Socket* with the JVC server address and specific port. Then it gets the *DataInputStream* and the *DataOutputStream* object over this *Socket*. The *Manager* sends the appropriate control message to the server through *DataOutputStream* when the participant issues the command to send the media stream or stop the sending through *control panel*. The *Manager* receives the result of request from the JVC server through *DataInputStream* and notifies the *Sender* of the result.

- *Sender*

The *Sender* captures the live audio and video streams from camera and microphone, sets up the media format and transmission parameters for captured audio and video

streams, and controls the transmission process by using the services of the JMF RTP manager.

The *Sender* implements the appropriate interfaces and callback methods of JMF RTP API. The operation of *Sender* consists of the following steps:

1. Join the multicast group with the specific multicast address and port. Audio and video streams use the same multicast address but different UDP ports. This is done by creating two *SessionManager* objects, which are used to send the audio and video stream respectively.

2. Specify the *MediaLocator* of the physical capture device. For example, the video capture card can be specified as "VFW://0" and the audio capture card can be specified as "javasound://11025".

3. Create the *Datasource ds1* from the *MediaLocator* to present the media data captured from this device.

4. Create the *Processor p* with the *ds1*.

5. Get the *TrackControl* objects from *p*.

6. Set track format of *p* with an RTP-specific format, such as JPEG_RTP or H263_RTP.

7. Set the content of *p* as RAW_RTP and realize the *p*.

8. Get the *Datasource ds2* from the output of *p*

9. Create the *SendStream* from *ds2*.

After the *SendStream* is created, the sending of audio or video can be controlled by starting or stopping the correspond *SendStream* and *Processor*.

- *Receiver*

The *Receiver* receives the audio/video stream through RTP data packets and the participant information through RTCP packet distributed in multicast group. As the *Sender*, the *Receiver* implements the appropriate interfaces and callback methods of JMF RTP API.

Receiving RTP streams is achieved by implementing the JMF *ReceiveStreamListener* interface. All events related to receive RTP streams will be posted through this interface. When the *NewReceiveStreamEvent* is received, the *stream* associated with this event will be extracted. Then the *Receiver* creates a *player* for this stream. The *Receiver* must register as a *ControllerListener* of this player, so that it can get notification when this player is realized.

When a *RealizeCompleteEvent* event is received, the *Receiver* extracts the associated *player* of this event. Then it checks whether any stream of this participant has already been rendering or not. If not, it creates a new instance of the *Render* as the GUI to render all streams of this participant in the future. If yes, there are two cases: If the current stream is audio, it gets the *GainControl* from the *player*. The *GainControl* controls the audio volume. If the current stream is video, it changes the corresponding render GUI mode to show the video.

The *Receiver* keeps track of all received streams and the associated *players* grouped by participant. By stopping or starting the corresponding *player*, the *Receiver* can pause or resume the receiving of the streams from specific participant respectively.

Beside stream data, *Receiver* also receives the participant information. In JMF, the participant information can be received from the following events by implementing the appropriate interface:

- TimeoutEvent, ByeEvent from *ReceiveStreamListener* interface: a participant leaves the session.
- NewParticipantEvent from *SessionEvent* interface: a new participant joins the session.
- ReceiverReportEvent, SenderReportEvent from *RemoteListener* interface: a participant is in active or passive status.

When the *Receiver* receives the participant information, it posts the participant information through an interface so that the interested objects, such as the *Manager*, can implement this interface to get the participant list.

Renderer

The *Renderer* provides a GUI to render the received streams. The media streams from different participants will be rendered in independent windows. The user can control the render activities with the control bar of the render window. The user control message, such as pause to receive video stream of a specific participant, will be sent back to the *Receiver*, which will search the corresponding *player* of this video stream and then do the appropriate action on this *player*.

Control panel

The *Control panel* is the graphics user interface for a user to access the audio-video service. It is composed of the control buttons and participants list. A participant can join or leave the audio/video session, and pause or resume to send his audio or video stream to other participants by clicking the corresponding buttons. The currently active and inactive participants will be listed separately so the participant can know who is in the section.

4. 4 Implementation

According to the JVC architecture and design presented, we implemented a JVC prototype with JDK1.2 and JMF2.0 at the MCRLab, University of Ottawa. The system platform of JVC consists of PCs connected through 100 M Ethernet. All PCs are Pentium 333 with 128 MB memory. The network operating system is Windows NT with service pack 5.

JVC is implemented with Object Oriented Programming. The main classes , interfaces will be introduced in the following section. We will also show how to use JVC in the last session.

4.4.1 Implementation of the JVC server

The figure below shows the high level UML[42] class diagram of the JVC server.

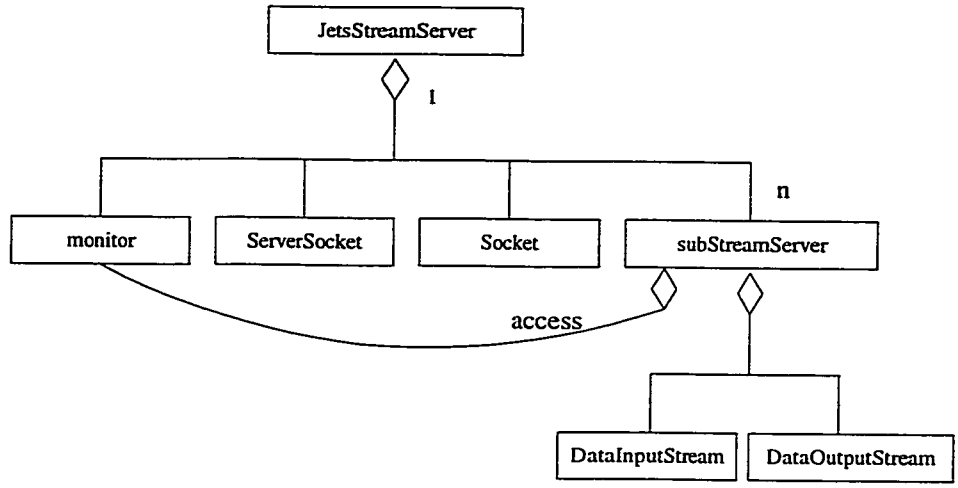


Figure 4.3 high level class diagram of the JVC server

The main classes are:

- *JetsStreamServer*

JetsStreamServer is a stand-alone Java application. It is implemeted as a thread. The *run()* method waits for a client connection and spawns a new *subStreamServer* object.

- *subStreamServer*

subStreamServer is implemented as a thread. The *run()* method reads the client's message from the DataInputStream object and delegates it to *handleSignal* method which will implement the admission algorithm.

- *monitor*

The *monitor* defines the synchronized methods to access the *token*.

4.4.2 Implementation of the JVC client

The figure below shows the class diagram of the JVC client:

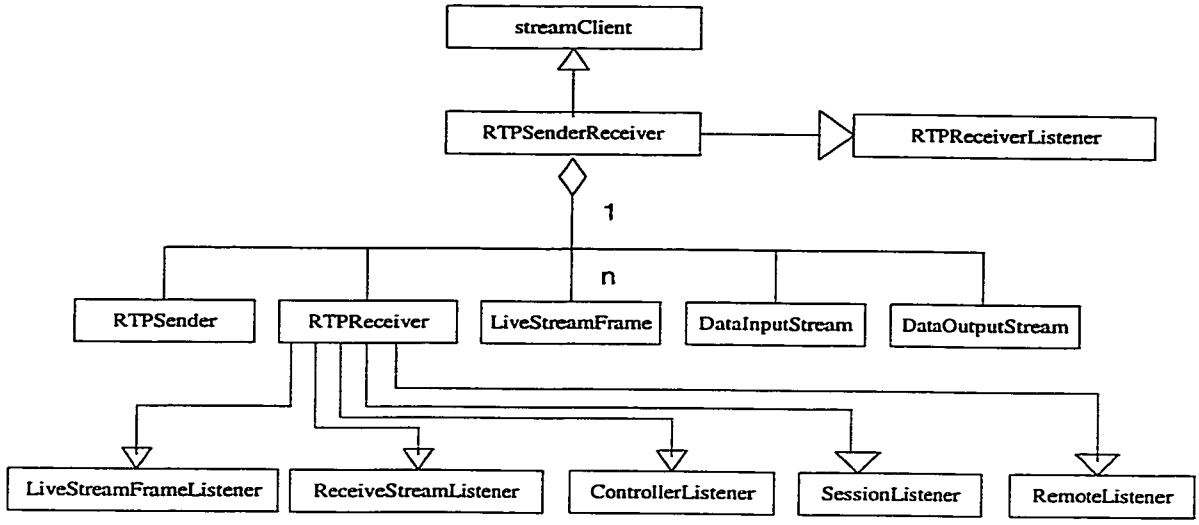


Figure 4.4 class diagram of the JVC client

The main classes include:

- *streamClient*

The *streamClient* creates a TCP/IP connection to JVC server and provides the *token* request methods which can be used by its subclasses.

- *RTPSenderReceiver*

The *RTPSenderReceiver* extends the *streamClient* and implements two interfaces: *ActionListener* and *RTPReceiverListener*.

- *RTPSender*

The *RTPSender* is implemented as a thread. It can capture and send both audio and video streams.

- *RTPReceiver*

The *RTPReceiver* is implemented as a thread, so it can run concurrently with the *RTPSender* to improve the JVC performance. *RTPReceiver* implements the *LiveStreamFrameListener* and the *RTPReceiverListener* interfaces. It also implements the JMF *ControllerListener*, *SessionListener*, and *RemoteListener* interfaces in order to be notified about JMF RTP events.

- *LiveStreamFrame*

The *LiveStreamFrame* implements the render GUI for received video and audio streams.

- *RTPReceiverListener*

The *RTPReceiverListener* is the interface that gets notifications of the participant information posted by the *RTPReceiver.*

- *LiveStreamFrameListener*

This is the interface for handling user interactions with the *LiveStreamFrame*, such as closing participant's audio and video streams.

4.4.3 Operation

JVC is very easy to use. First, the user should input the RTP multicast address, the audio and video ports used for the live audio/video session. Then the JVC control panel will appear, as shown in the following figure [fig 4.5].

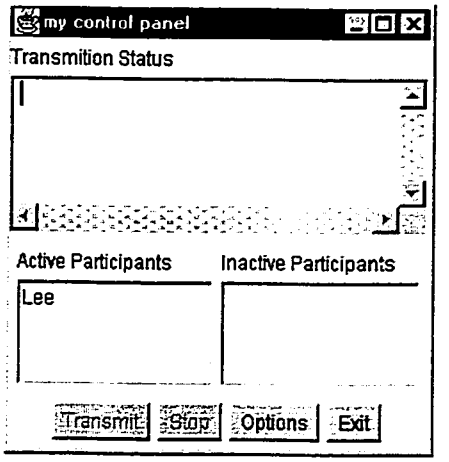


Figure 4.5 The control panel of the JVC client

When JVC receives a new audio or video stream from a remote participant, it will present the stream(s) in an independent window, as shown in the following figure[fig 4.6].The rendering of video and audio stream can be paused or resumed by clicking the associated Video and Audio button of this window. Audio volume can be controlled by the "Volume" button.

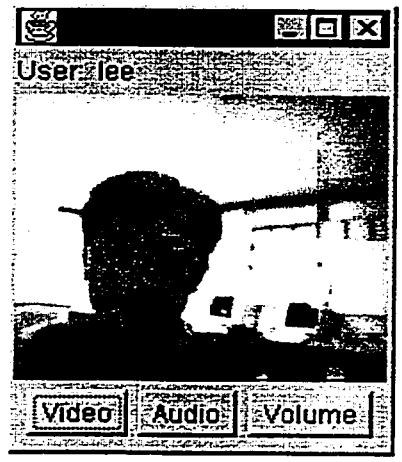


Figure 4.6 presenting the remote participant audio / video stream

JVC can work with the JETS whiteboard. In a collaborative session, the user uses JVC to transmit live audio/video and the JETS whiteboard to provide a shared workspace.

Chapter 5 Design and Implementation of JVCR

In this chapter, we will present the design and implementation of Java-Enabled Videoconferencing Recorder Tool(JVCR). We first introduce some key design issues of JVCR in section 5.1, then we present the architecture and components of JVCR in section 5.2, JVCR implementation will be presented in section 5.3 . Finally, we introduce the operation of JVCR in section 5.4.

5.1 General specification

In this section, we will summarize the goals and key design decisions of JVCR in high level.

5.1.1 Goals

The overall goal of JVCR is to enable effective asynchronous collaboration by providing session archive service. In order to achieve this goal, some subgoals have been identified:

Unobstructive recording

The JVCR should be an "added-value" feature supporting the natural activities of the collaborative session. The unobstructive recording means a transparent recording to the recorded application and the participant (except the scribe) in this thesis. It is a key goal based on the following reasons:

- In the Web-based collaboration system, different applications, including JVCR, are loosely coupled together to provide an integrate solution for collaboration. Generally, these recorded applications are "recording-unaware". They should not change behaviors in order to be recorded, although this will greatly simplify the JVCR design. From the development point of view, this means that any source code of other systems should not need to be changed.

- The live session quality should never be degraded by JVCR.
- JVCR should not constrain the behaviour of participants of the live session. They can do whatever they want as if the JVCR were not running.

Interactive recording

JVCR should provide interactive control ability so that people can customize it to satisfy the specific requirement. For example, people may specify the options such as:

- media type(s): which media streams need to be recorded
- session time segment: when to begin recording and when to stop it.
- location in the network: where the recording takes place.

To be more accessible, JVCR should provide a platform-independent, intuitive and easy-to-use graphics user interface for the users to issue their recording request and control command.

Integrated service

A Web collaboration system may consist of applications with different architectures, service interfaces and media broadcast approaches. For our project, we have two different components: JETS shared applets and JVC. JETS use an event broadcast method to achieve collaboration. The instance of shared applet in each site connects with the JETS server through TCP/IP. All events will be sent to the JETS server first, then the JETS server broadcasts this event to other sites (an exception is the events request for floor control, which will not be broadcast). JVC, on the contrary, uses RTP over MBONE to transmit the audio and video streams. Every participant can receive the streams if he joins the multicast group. Therefore, JVCR should provide an integrated service interface so that it can work with different collaboration applications and hide all these differences to the participant.

Open format

In order to support interoperability, JVCR should store the recorded session with application-independent data format so that it can be reused by the third-party system.

Finally, as a tool used in a Web-based collaboration system, JVCR should be simple and robust. It should not be too complex because this will prevent integration with other systems, as well as limit the evolution of the system and make it useless in the long run.

It is a challenging task to design a recording tool for the Web-based multimedia collaboration system because it involves several areas and some latest technologies. Now we introduce our high-level design decisions.

5.1.2 Design decisions

Generally, a complete recording process includes the following stages:

- Capture: capture the collaboration content.
- Handle: do the appropriate actions with the collaboration content.
- Store: save the handled collaboration content to disk so it can be used asynchronously.
- Playback: play back the recorded content. Strictly speaking, this is not included in the recording process. However, the playback approach has strong influence on design of recording service, so we consider it here.

Capture

As the first step of recording, JVCR needs to know what is happening in a session. A basic feature of a collaboration system is that every participant can receive all broadcast events and media streams in the shared space if he has the required

permission. Therefore, a straightforward approach is that JVCR joins the session as a special participant. The difference from other participants is that JVCR basically is a receiver without sending out any media streams to be broadcast while other participants may be a sender as well as a receiver at the same time [fig 5.1].

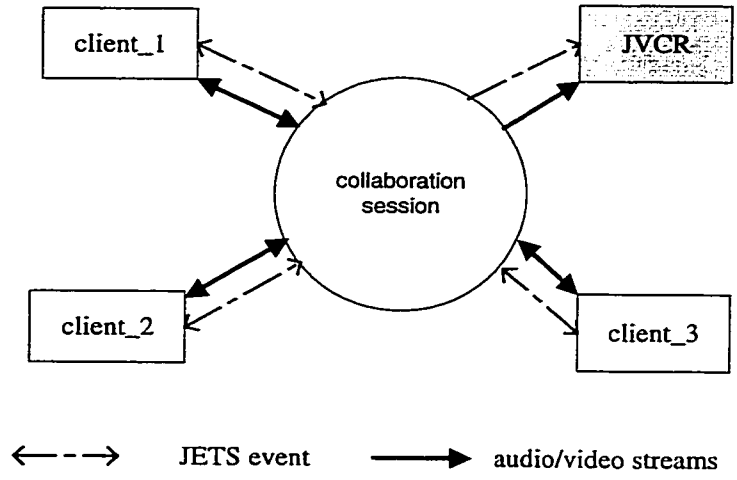


Figure 5.1 JVCR joins the session as a participant

As we mentioned already, JETS applications and JVC use different mechanisms to achieve collaboration. JETS uses an event-sharing approach so that every participant has the What-You-See-Is-What-I-See view. JVC distributes the live video and audio streams among all multicast group. To work with both of them, there are two possible ways:

- Use one recorder to capture both JETS events and JVC streams. This central processing model is easy to design and implement. A drawback is that JVCR may suffer performance problems because one recorder can not respond in a timely way in the case of a heavy load.
- Use two recorders, one for JETS applications and another for JVC. This approach separates the different tasks logically. Performance will be better because the two recorders can work concurrently. Furthermore, different recorders can be loaded dynamically according to the user's requirement that

decreases the workload of the server. However, we need a coordinator to coordinate the two recorders properly.

To achieve high performance, JVCR uses two recorders.

The capturing is controlled explicitly by the user who has been granted suitable permission to do so. The user can start, pause, resume and stop the capturing at any time.

Data Handling

The captured data can be divided into syntactic information and semantic information. Syntactic information is system-level data and is application-independent, such as raw audio and video packet. Semantic information, on the other hand, is application-dependent, such as a JETS event broadcast in a collaboration session. There are three possible handling approaches for a recording system based on the levels of awareness for how it interprets captured data [25].

Approach	Awareness of captured data
Application Independent	unaware
Application Dependent	Semantically aware Syntactically unaware
Application Dependent	Fully aware

Table 5.1 Three possible handling approaches

- *Unaware*: the recording system does not interpret the contents of captured data, but adds some tags to the data so they can be differentiated. When played back, captured data is sent back to the corresponding application, which is in charge of interpreting and playing back the data. This architecture is most flexible because it is transparent to recorded applications. The drawback is that the captured data format is application-dependent.
- *Semantically aware/syntactically unaw*are: the recording system fully interprets all semantic information, but does not interpret syntactic information. It is

application-dependent bacause the recording system needs to know the application semantics.

- *Fully aware*: the recording system interprets all captured data. The advantage is that the captured data is application-neutral and self-explained. However, the recording system must know the recorded application context and process logic.

One key goal of JVCR is to create application-independent archive data. Therefore, JVCR uses a fully aware architecture. So the question is how to extract the multimedia objects from the JETS events and audio/video streams so that they can be stored with well-known formats.

The media object is defined according to the media type. For live audio and video streams, a media object is defined as each continuous segment of live audio and video.

For the JETS whiteboard, there are two different kinds multimedia objects involved in the JETS events:

- Pre-existing media files which are loaded to the collaborative space from disk. There are two kinds of such files used in the JETS whiteboard. One is discrete media, such as image. Another type is continuous media, such as video clip and audio clip.
- Dynamic collaborative information, which is created by users' interactions, such as an annotation drawn by the users in the whiteboard, or live chat text input by users.

In the first case, the loaded file is the media object. In the second case, we define the dynamic information created in a continuous interaction as a media object. For example, an annotation drawn from the mouse click to mouse up is a media object.

Because JVCR interprets all captured data, it should have knowledge of the events broadcast by the JETS server, which is predefined by the application developer. In other words, JVCR needs an application dependent event handler that knows the semantics of events of specific implementation of the JETS whiteboard. Whenever JVCR captures an event through the event receiver, it will delegate the event to the event handler. The event handler is somewhat similar to the JETS whiteboard, but without the display window. It interprets every JETS event in the same way as the whiteboard in order to keep track of the whiteboard's current state.

The event handler will explain each event. The goal is to identify the multimedia objects used in the event. A general process to generate a media object from a JETS event is given as follows [fig 5.2].

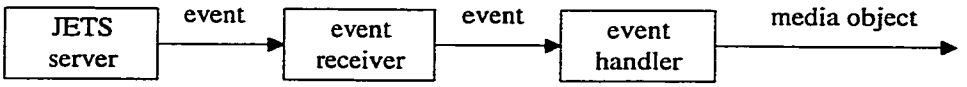


Figure 5.2 a general process to extract a media object

As we can see, the event handler does not connect to the JETS server directly and the event is passed by the event receiver. The separation of the event receiver and the event handler can provide a more flexible plug-in architecture, so that different event handlers can be plugged into JVCR according to well-defined interfaces. This can be done by designing an abstract class as a template of the event handler. The abstract class defines the common interfaces and methods but leaves them to be implemented by each specific event handler according to the semantics of different sets of events. The class structure model is given in the following figure[fig 5.3]:

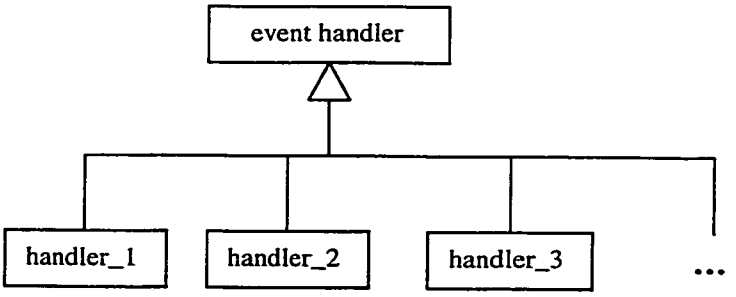


Figure 5.3-event handler class

The event handler should be dynamically found and loaded by JVCR through its name registered in the configuration file using Java's dynamic loading mechanism.

JVCR should preserve the temporal relationship among different objects. JVCR uses timeline mode and creates a timestamp for every event or stream it receives. The issue here is who provides the timestamp and how to count it. As we mentioned above, JVCR is designed to work with recording-unaware applications, which means JVCR does not need to change any behavior or source code of the original applications. However, we can not guarantee that the event broadcast by the original applications will carry timestamp information. Therefore, JVCR will generate the timestamp for the received event. Because the recorder for JETS and recorder for JVC are running in the same machine, it is convenient to synchronize the recorded JETS event and media streams.

The timestamp is a relative value, counting from the system clock. As we mentioned before, JVCR loads the recorder on demand in order to keep server workload lower. However, this initialization process causes some latency, especially for audio-video recording. In order to skip this empty time gap, the beginning timestamp of recording process should be defined as the system clock time on the server when the first JETS whiteboard event or JVC audio/video stream is received by the JVCR server.

With the help of timestamp, we specify the "*active time*" of each media object. The *active time* is the time from when it is loaded to when it disappears in the collaborative space. The synchronization relationship among different media objects can be preserved by specifying the *active time* of each media object.

For example, in time t1, JVCR receives event1, which loads *img1.jpg* into the whiteboard; and in time t2, JVCR receives event2, which load another image to replace *img1.jpg*. Then, the *active time* of img1.jpg is t2-t1. [fig 5.4]

t1: Event1: load image img1.jpg in Whiteboard
t2: Event2: load another image to replace img1.jpg
t2-t1: the active time of img1.jpg

Figure 5.4 an example of counting active time

By specifying the "*active time*" of a media object, it is easy to explicitly specify the time slot of every media object occurring in the session, even for media without intrinsic time such as images and text. On a given time slot, there may be serveral media objects.

JVCR creates a SMIL document to combine all media objects together by specifying the synchronizaiton relationship among media objects.

Store

The captured media objects and associated temporal relationship should be saved in a repository for later playback. JVCR stores the session in a database and a file system. The database stores the temporal relationship and the links for all captured media objects. The dynamically created media objects themselves are stored in the file system. The record of database can point to the file through its link which is the URL of the file. The URL gives a global unique name which can be found over the Internet.

In order to be flexible and not be bound to a specific kind of database, JVCR accesses the database through the Java Database Connect (JDBC) API. JDBC consists of a set of classes and interfaces written in Java and provides database access that is independent of both the platform and the database host system that the application runs on. JDBC enables the developer to write Java code that establishes a connection

to a Structured Query Language (SQL)-capable data source, sends SQL statements to the data source, and returns status messages and data records resulting from the SQL execution. JDBC supports two kind of database access models: two-tier model and three-tier model.

In the two-tier model, a Java applet or application accesses the database directly. This requires a JDBC driver that can communicate with the particular database being accessed. A client's commands are delivered to the database, and the results of those statements are sent back to the client. The database may be located on another machine to which the client is connected via a network. [Fig 5.5]

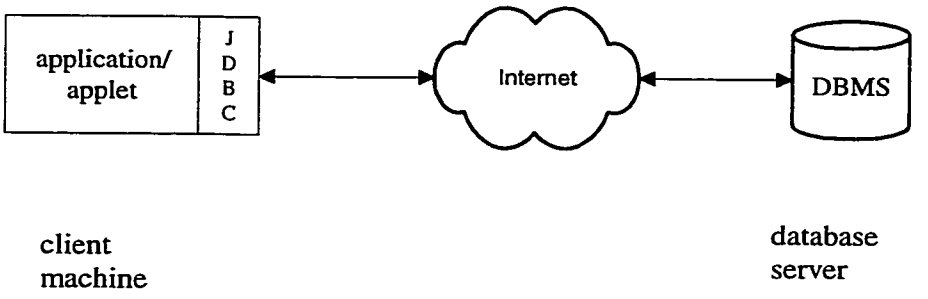


Figure 5.5 the two-tier Database Access Model

JVCR uses a three-tier database access model. The three-tier model consists of three tiers with different responsibilities: the client tier implements the user interface; the middle tier implements all business logic and database access; and the third tier is the database itself. In the three-tier model, user commands are sent to a "middle tier" of services, which then sends the commands to the database. The database processes the commands and sends the results back to the middle tier, which then sends them to the client. The three-tier model makes it possible to maintain control over access and the kinds of updates that can be made to database. Furthermore, the three-tier model is easy to deploy and maintain.

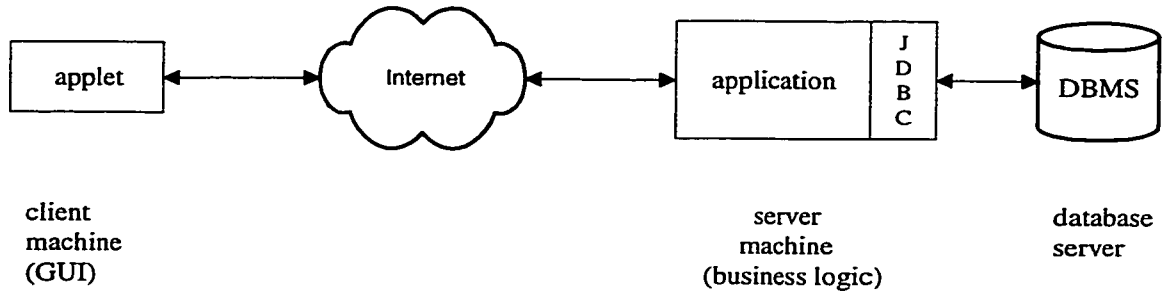


Figure 5.6 the three-tier Database Access Model

playback

Collaboration sessions are recorded as SMIL documents, so they can be played back by SMIL players. To facilitate access to the recorded sessions, JVCR also creates a recorded session catalogue which is an HTML page including the hyptertext links to the SMIL document of the recorded session. Users can automatically link to that page by a mouse click in JVCR Web-based user interface or inputing the catalogue page's URL directly in the Web browser.

The recorded session is available for playback immediately, so that it can be played back concurrently with the live session. This is a very useful feature for the latecomer to a live session. By reviewing what happened several minutes earlier, he can catch up with the ongoing session easily.

The playback process should be easy to use. One approach is to set the SMIL player as a Web browse plug-in or a helper application. When the user selects a session from the catalogue page, the session will be played back automatically.

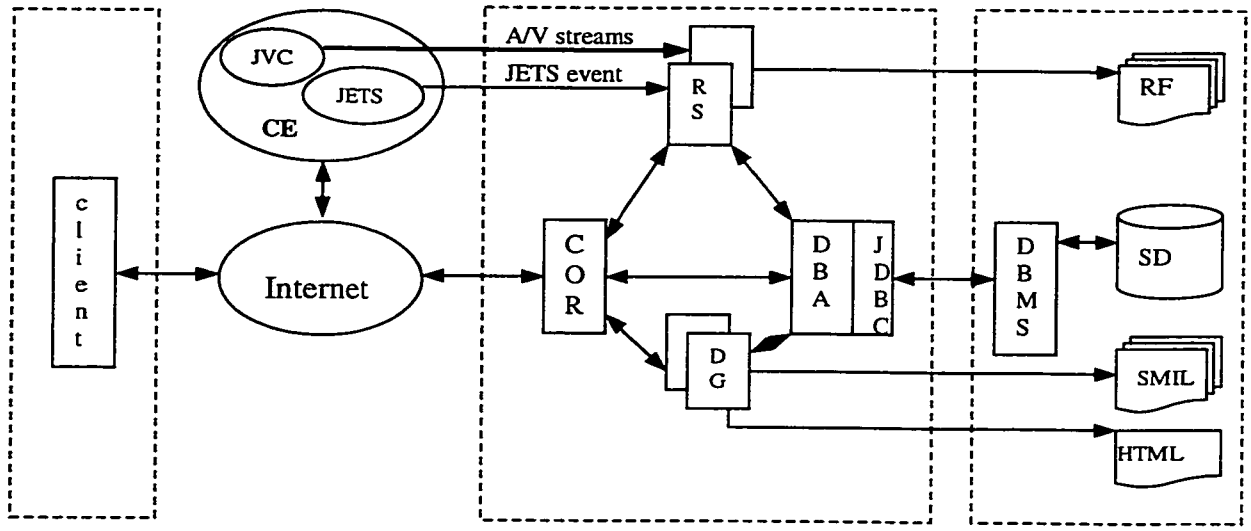
5.2 Design of JVCR

5.2.1 System Architecture

With system goal and high-level design decisions in mind, we propose JVCR architecture. JVCR is a client-server system with three parts:

- JVCR client
- JVCR server
- Data repository

The following figure shows the architecture of JVCR. [Fig 5.7]



CE: collaboration environment
RS: a set of recorders
COR: coordinators
DG: a set of multimedia document generators
DBA: database access wrapper
RF: resource files
SD: session information database
SMIL: SMIL document
HTML: HTML document

Figure 5.7-system architecture of JVCR

The *JVCR Client* provides a graphics user interface to access JVCR services. It is an applet and can be run in any Java-compliant Web browser. The JVCR client will be automatically downloaded to the user machine.

The *JVCR Server* provides recording service for the collaboration session. It has four major components.

- COR (Coordinator): the component to coordinate the behavior of all other components of the JVCR server. All user requests will be sent to COR first. It then dispatches the user requests to other components.
- RS (Recorders): a set of recorders that capture, handle and store the session data. RS consists of the JETS recorder *wbRecorderServer* and the JVC recorder *streamRecorderServer*. The JETS recorder connects with the JETS server through TCP/IP while the JVC recorder joins the JVC multicast group through RTP over MBONE. RS saves the dynamic collaboration information as files. In the meantime, it stores the extracted media object link and active time to a database.
- DG (Document Generator): a set of document generators. DG consists of the *smilCoder* and the *catalogProducer*. The *smilCoder* generates SMIL documents and the *catalogProducer* creates an HTML catalogue page for recorded sessions.
- DBA (Database Access Wrapper): the database access interface between the JVCR server and back-end database. It is a thin wrapper of JDBC that hides JDBC API from the database service clients. All components access database through DBA, which provides a unified and easy-to-use interface.

The *Data repository* manages the database and file system used to store the JVCR archive data. It includes the following components:

- RF (Resource Files): These resource files are dynamically created to store media objects. These resource files are application independent with a well-known format. For example, an annotation will be saved as JPEG file; audio stream will be saved as the AU files.

- SD (Session Information Database): a SQL database that saves the links of media objects and their *active time*.
- SMIL documents: the documents that specify the presentation of recorded sessions with SMIL language.
- HTML document: the document that lists all recorded sessions with the hyperlink to their SMIL document.

The *Data repository* and JVCR server may run at different hosts.

The operation of the JVCR is controlled by the users. The following is a typical scenario to record and play back a session:

1. User inputs the URL of the JVCR client applet in the Web browser.
2. JVCR client applet is downloaded from the Web server automatically to the user's machine.
3. User inputs the session description metadata and customizes his recording, such as specifying which application should be recorded.
4. User clicks on the appropriate button to send the recording request.
5. COR receives the request and analyzes which recorder (or both recorders) should be loaded.
6. If user requires recording the JETS whiteboard, COR will load the JETS recorder. The JETS recorder then joins the JETS whiteboard automatically and begins to record the events. If user requires recording JVC audio/video streams, COR will load the JVC recorder. JVC recorder then joins the JVC multicast group and begins to record the audio/video streams.
7. SD will be updated, and RF will be created by recorders.
8. User clicks on the appropriate button to stop the recording request.
9. COR receives the request and stops the loaded recorder.
10. User clicks on the appropriate button to create an SMIL document of the recorded session.

11. COR receives the request and forwards the request to the SMIL generator.
12. SMIL generator searches the SD and creates the SMIL document.
13. User clicks on the appropriate button to create HTML index document of the recorded session.
14. COR receives the request and forwards the request to the HTML generator.
15. HTML generator searches the SD and creates the catalogue page for all recorded sessions.
16. User goes to the catalogue page and selects the recorded session to play back.

5.2.2 Design of the JVCR server

In this section, we give more details of the JVCR server design.

5.2.2.1 COR (Coordinator)

COR provides a unified service interface to clients. It is the coordinator component of the JVCR server. COR is composed of the *RecordManager* and the *subRecordManager.*

RecordManager

The *RecordManager* is a thread dispatcher. It initializes the server socket and waits for client connection. When a client connects to the JVCR server, RecordManager spawns a new *subRecordManager* object to serve this client. Furthermore, *RecordManager* also creates a new client socket for the new *subRecordManager* instance so that each instance of *subRecordManager* can communicate with client using a separate socket.

The advantage of this is that multiple *RecordManager* instances can run concurrently which will maximize the real-time behavior. The procedure is shown in the following figure [fig 5.8]:

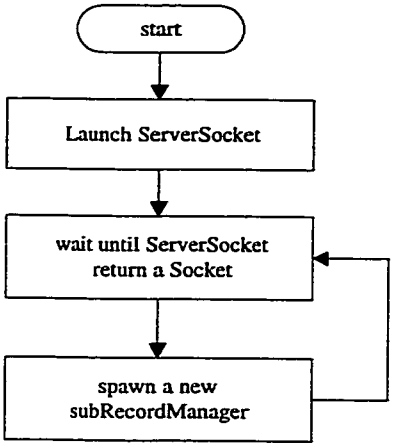


Figure 5.8 spawns a subRecordManager

subRecordManager

The *subRecordManager* is the object that actually works as a coordinator. The responsibilities of *subRecordManager* are:

1. Receives all requests from JVCR clients and returns the execution states back to them.
2. Launches other server components on demand after it receives a client request.
3. Dispatches the user requests to corresponding server components.
4. Listens to other server components state changes through announcement / receiving machanism.
5. Saves the session metadata such as sessionId, sessionTitle, recording time, etc. to the session catalogue database. However, *RecordManager* doesn't handle any JETS events or RTP packets itself. On the contrary, it delegates this task to *wbRecorderServer* and *streamRecorderServer.*

The *subRecordManager* gets the client request as well as sends back the feedback to the client through the DataInputStream and the DataOutputStream object respectively. Both of them are passed from the *RecordManager.*

In order to dispatch the client request to the appropriate objects as well as get the status of other JVCR server components, *subRecordManager* has a bi-directional communication with these objects, which is designed according to the Observer pattern[39]. Each recording server and document server posts its events through a well-defined interface. The object can register as a listener of these events and get informed when an event is posted. Therefore, the bi-directional communication is achieved with the following approach:

1. *subRecordManager* registers as the event listener of recording servers and document servers.
2. recording servers and document servers register as the event listener of *subRecordManager.*

Whenever *subRecordManager* receives a client control command, it posts this control command through its interface.

The recording servers are launched dynamically. Besides the consideration of decreasing the workload of the machine running the server, the major reason is to dynamically configure the recording server. For the *wbRecorderServer*, the parameters of JETS Whiteboard applet are embedded in an HTML page and not stored in the server. For the *streamRecorderServer*, the multicast address, audio and video RTP port, and ttl may be different for different recording clients. In both cases, the JVCR client can specify these parameters and send them to the JVCR server. The *subRecordManager* then initializes and loads the corresponding recording server with these parameters.

When *subRecordManager* receives a new recording request, it means a new recording session begins. *subRecordManager* then creates a new record in database and creates the directories for storing the media objects captured in this session. The flowchart below [fig 5.9] shows the operation of the *subRecordManager.*

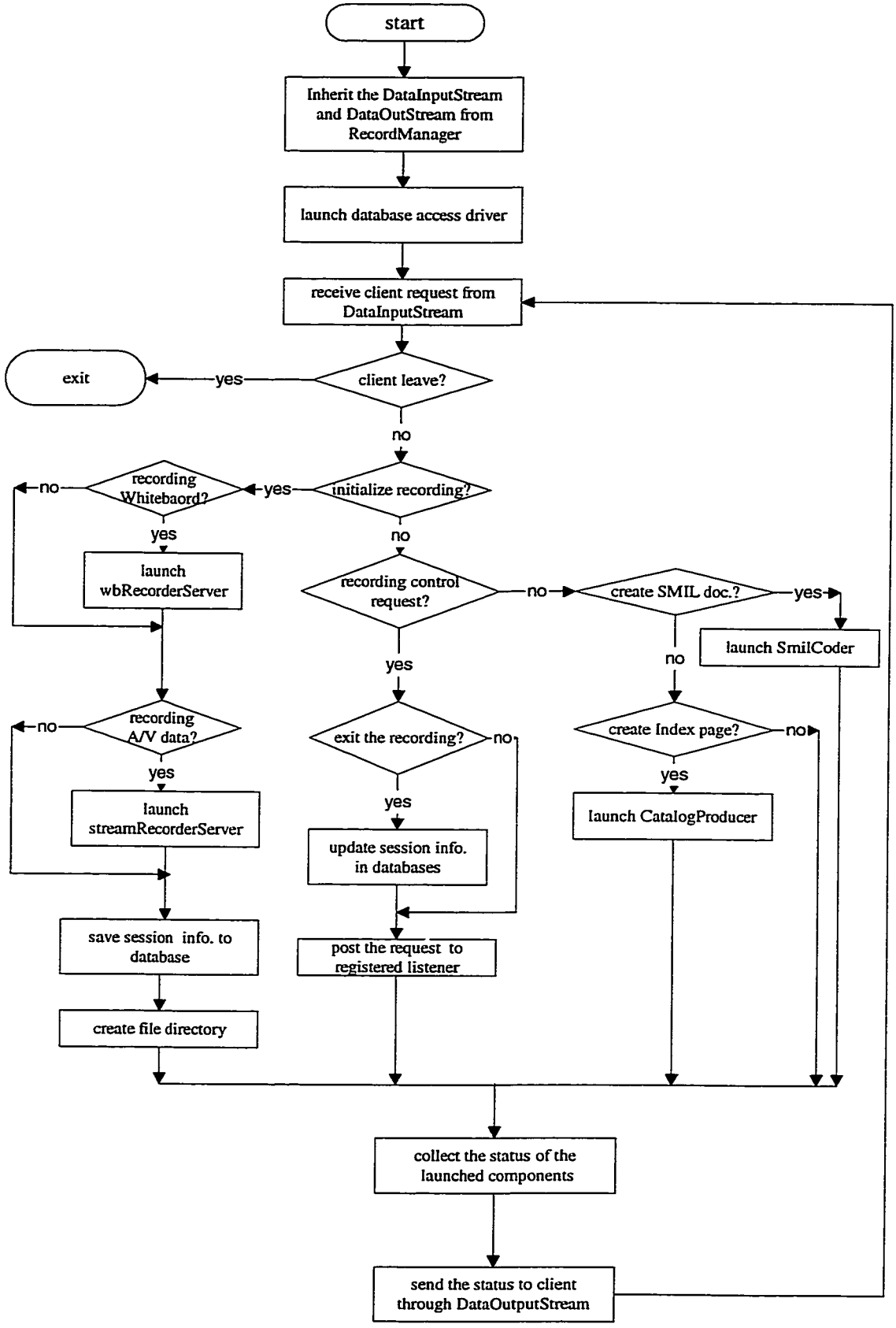


Figure 5.9 the operation of subRecordManager

5.2.2.2 RS (Recorders)

RS provides recording service. It consists of the *wbRecorderServer* and the *streamRecorderServer*.

wbRecorderServer

The *wbRecorderServer* records user interaction of the JETS whiteboard. Its responsibilities includes:

1. Connects to JETS server through TCP/IP Socket connection.
2. Receives the JETS events broadcast by the JETS server.
3. Handles every JETS event and extract media object.
4. Saves the media object link and active time to a database.
5. Saves the dynamic collaboration information as files.
6. Notifies the state to registered listeners.
7. Registers as a *subRecordManager* state listener and do appropriate action when it is informed of the state change.

wbRecorderServer consists of an event receiver and an event handler. The following figure [fig 5.10] gives the initialization operation of the *wbRecorderServer*.

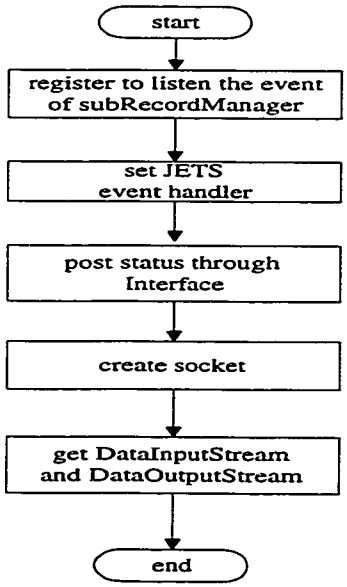


Figure 5.10 initialization operation of *wbRecorderServer*

The JETS event handler is dynamically found and loaded through its full class name.

The *wbRecorderServer* works as a special JETS whiteboard client. It never requests to use the shared resource because the only goal it joins the whiteboard for is to listen to what the JETS server broadcasts. Moreover, it runs on the server side and saves the shared media object. It does not render them in a whiteboard user interface.

The *wbRecorderServer* uses the same approach to receive a whiteboard event as other whiteboard clients. From the socket connecting with the JETS server, *wbRecorderServer* gets I/O streams for JETS data channel and signaling channel to listen for a JETS event. Then, the event is timestamped and delegated to the event handler to extract the media object and its active time.

The media object is specified by its URL. For a pre-existing media object, the URL of the file is created by combining the file path and file name. The file path is sent by JVCR client. The file name can be obtained directly because it has already been included in the JETS event message. For example, the following event is to load an image to whiteboard:

2	image001.gif

Figure 5.11 an example of a JETS event

For a dynamic media object, a file will be created to store this object with application-independent standard format. For example, the annotation with its background will be saved as a JPEG file. Then, the URL of this file is used to specify the dynamic object.

The *wbRecorderServer* registers as the user control command event listener of the *subRecordManager*. It can start, pause, resume or stop recording according to the control command.

streamRecorderServer

The *streamRecorderServer* records JVC audio and video streams. The responsibilities of *streamRecorderServer* include:

1. Joins the corresponding JVC RTP stream session(s) as a passive participant.
2. Saves the media object link and *active time* to a database.
3. Saves the audio / video streams to the disk files.
4. Notifies the state to registered listeners.
5. Registers as a *subRecorderManager* state listener and does appropriate action when informed of the state change.

streamRecorderServer makes use of JMF RTP API to receive and store the audio/video RTP streams. The following figure [fig5.12] gives the operation when *streamRecorderServer* receives notification of a JMF *ReceiveStreamEvent.*

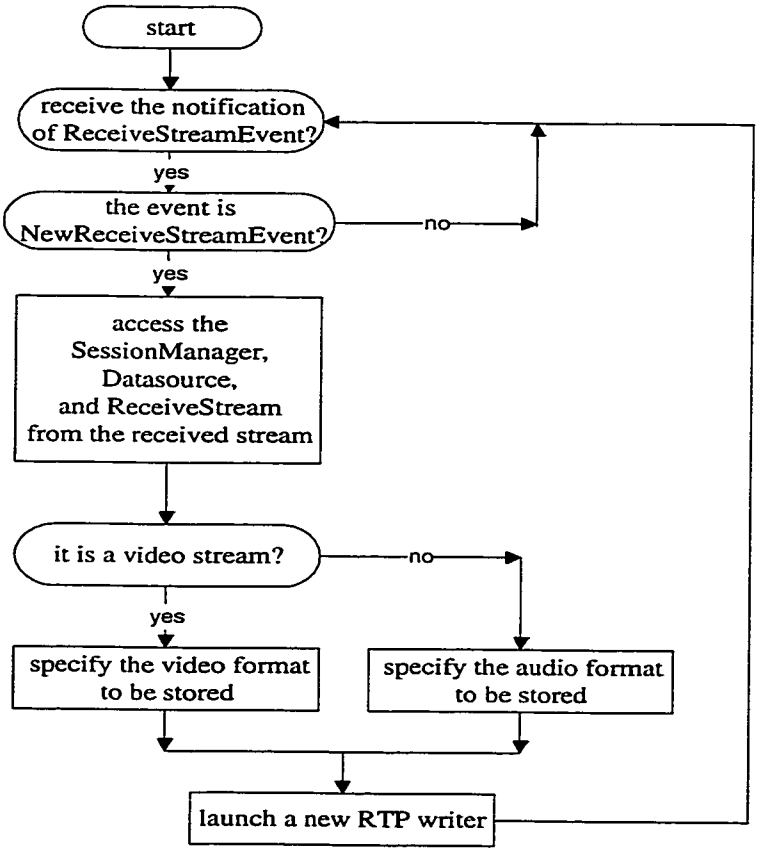


Figure 5.12 the operation of a *streamRecorderServer*

In order to receive audio or video RTP streams, *streamRecorderServer* implements the interface *ReceiveStreamListener* of JMF RTP API. Therefore, it will receive notification whenever:

- New receive streams are created.
- The transmission of data starts or stops.
- The data transmission times out.
- The received stream's format or payload changes.

There are seven events posted by JMF through this interface, including the event *NewReceiveStreamEvent,* which indicates a new stream has been received. When *streamRecorderServer* gets notification of this event, it can not only access the stream but also the datasource and participant who sends this stream. Then it launches a new *RTP writer,* which is a thread to write the detected RTP stream to disk as a separate file. *RTP writer* creates *Datasink* and *Processor* object from the stream and uses them to control the recording:

- Starts recording: open and start the *Datasink*, then start *Processor.*
- Pauses recording: stop the *Processor.*
- Resumes recording: start the *Processor.*
- Stops recording: stop and close the *Processor*, then close the *Datasink.*

As the *wbRecorderServer,* the *streamRecorderServer* listens to the events posted by *subRecordManager.* When it gets notification of the user control request, it controls the recording of the stream through the corresponding *RTP writer.*

5.2.2.3 DG (Document Generator)

DG consists of the *smilCoder* and the *catalogProducer.*

smilCoder

The *smilCoder* creates SMIL documents that specify the presentation of the recorded session by combining the recorded multimedia objects according to their temporal

relationship. Because the active time of media objects as well as their URLs have already been recorded in the database, the temporal relationship of all media objects can be easily specified.

The *smilCoder* specifies all media objects as the children of the *"par"* element. For each media object, the begin and end time-point will be specified explicitly according to its *active time* during collaboration. The *"fill"* attribute of each media object is specified as *"remove"*, because the media object will be removed as soon as the end timestamp is over. There are several important advantages doing this:

- efficiency. The SMIL document can be generated by scanning the database once.
- accuracy. One media object presentation's delay will not affect the others because each media object will be presented according to its own begin and end time point.
- ease of implementation.

However, it is very difficult to specify the screen layout in exactly the same way as that of the recorded application with SMIL. First, participants may interact with several shared applications at the same time. The GUI of these applications may have arbitrary complicated relationships. For example, the GUI of an active application may overlay with that of an inactive application. Another reason is that there may be no consistent screen layout among all participants. For example, a participant may change the size of the shared application's screen layout on his machine without affecting other participants. Therefore, the *smilCoder* simply combines all media stream presentation space together and divides the screen into three rectangular regions:

- whiteboard region: presenting all interactions on the JETS whiteboard except chat text.
- chat region: presenting all chat text input in the chat box of the JETS whiteboard.
- live audio and video region: presenting the audio and video streams.

The *"fit"* attribute of each region is specified as *"scroll"* to support the scrolling mechanism in case the region's rendered contents exceed its bounds.

All chat messages will be written to a text file first, then, this file will be specified as one media object element as the child of *"par"* element. The *"begin"* attribute of this media object will be specified as the begin timestamp of the first chat message. The *"end"* attribute will be specified as the end timestamp of the last chat message.

The flowchart below [fig 5.13] gives the operation of a *smilCoder:*

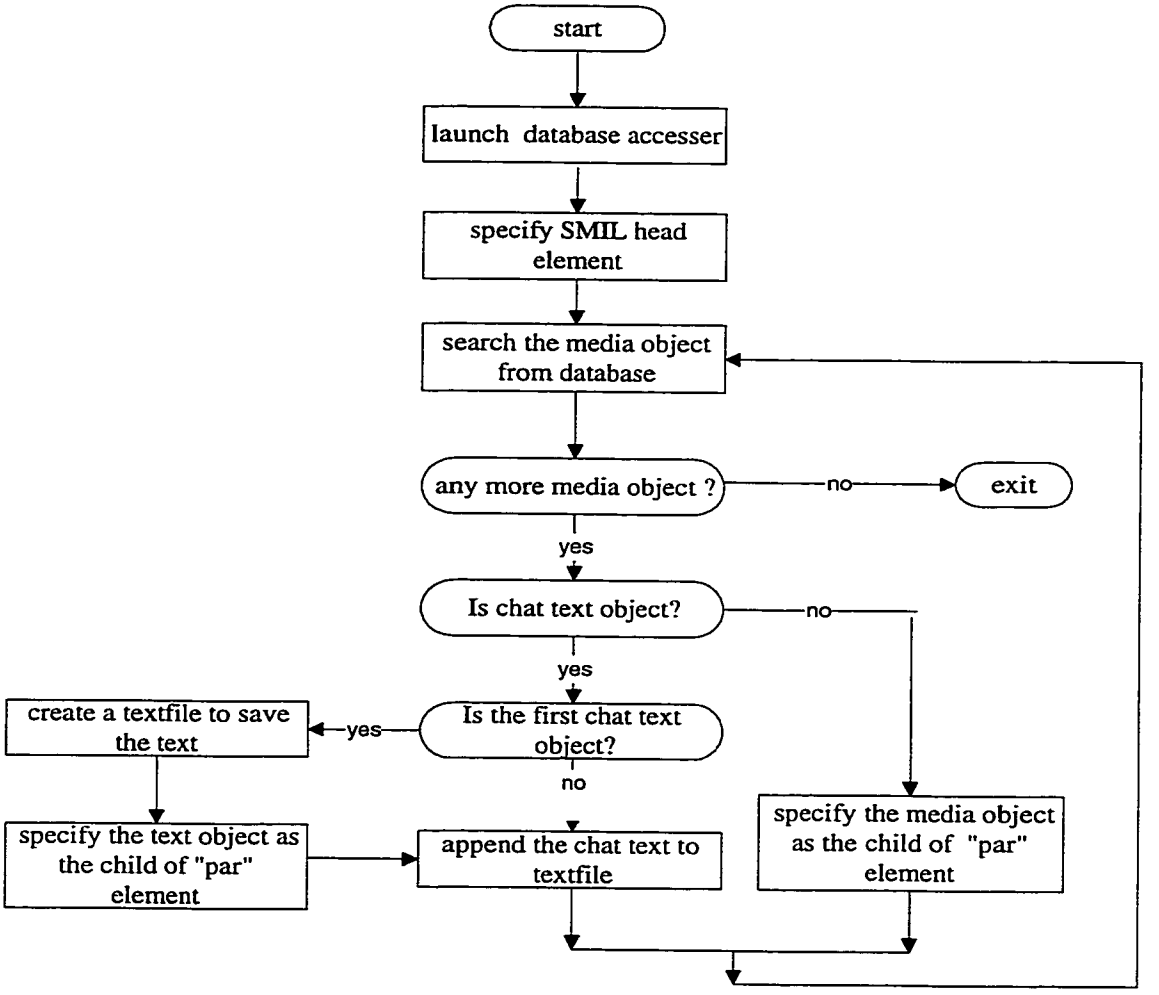


Figure 5.13 the operation of a *smilCoder*

catalogProducer

The *catalogProducer* creates an index file with HTML format for all recorded sessions. The HTML page includes the metadata of all recorded sessions specified by the recording user as well as the hypertext link to the SMIL document of the recorded session.

catalogProducer searches the session database and handles the database records one by one. The HTML page is written through *java.io.printerwriter*, which is used to write the text file.

5.2.2.4 DBA (DataBase Accesser)

The *DBA* is a thin wrapper of JDBC which hides JDBC API from the database service client as well as provides a unified and easy-to-use interface. The *DBA*:

1. Loads SQL database driver.
2. Creates the connection with the database.
3. Parses the client's parameters and creates SQL statement.
4. Forwards the SQL statement to the JDBC API, which will communicate with the database directly.
5. Combines the database search result and returns it to the client.

5.2.3 Design of the JVCR client

The JVCR client can be a JAVA applet or a Frame launched within an applet. It can run within any JAVA-enabled Web browser. Its main functionality is to provide the GUI to access the recording service as well as set the data channel to communicate with the JVCR server.

Communication between JVCR client and server goes through the TCP/IP socket. Once downloaded into the user machine, the client applet establishes a Socket connection to the JVCR server. From the Socket, it gets I/O Streams for

communication channel with server. The channel will be filtered from *InputStream* and *OutputStream* to *DataInputStream* and *DataOutputStream* to facilitate I/O operation for Java primitive data types.

The parameters of the client applet are embedded in the HTML page where this applet resides. These parameters include the JETS server address and Audio/Video multicast session parameters. The applet will send these parameters to the JVCR Server in the initialization phase of recording. The JVCR server then uses these parameters to initialize the whiteboard and audio / video recorders.

The message transmitted between JVCR client and server is packaged as a packet with the following format [fig 5.14]:

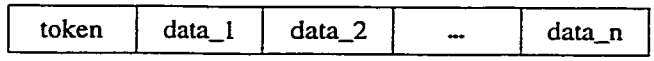


Figure 5.14 the packet format transmitted between the JVCR client and server

Each message is preceded by a *token* which indicates the semantics of the message, and is followed by its own set of data. The token is 1 byte, and each data_i has the built-in JAVA type with variable length.

The client and server communicate with three kinds of packets:

a. Initialize packet: sent by client to server, including the parameters to initialize a session recording. The token is 0.

b. User Control packet: sent by client to server, including the user control command. The token is 1.

c. Feedback packet: sent by server to client, including the return values and status of the user request. The semantics of the token used in this packet are:

- Token 10: available sessions which have not been created as SMIL documents yet.

- Token 11: the status of the SMIL document creation.
- Token 12: the status of the HTML page creation.
- Token 13: the status of the recording server.

The GUI consists of widgets to input the session metadata as well as the buttons to fire corresponding recording actions. The buttons can be divided into two groups: one for recording, another for generating the SMIL document and catalogue HTML page. The buttons should be enabled or disabled according to the current recording status to prevent misuse. The following figure shows the GUI [fig 5.15]:

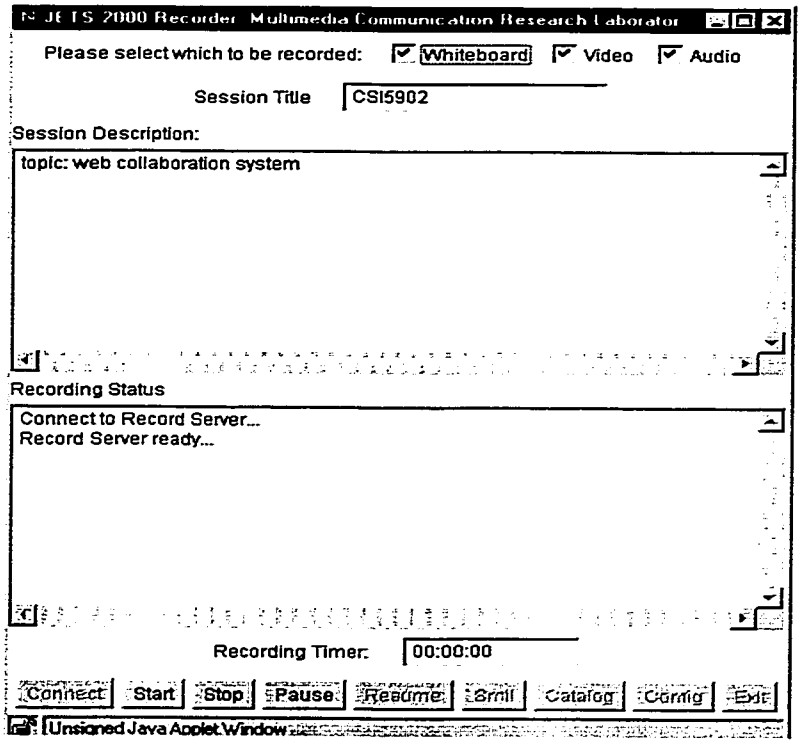


Figure 5.15 the JVCR client user interface

5.2.4 Design of the JVCR data repository

The JVCR data repository consists of an SQL-enabled relational database and the hierarchy file system.

The database stores the hypertext link and active time of each captured media object. The live audio / video streams, the annotation and chat text of JETS whiteboard are stored as files.

There are three tables defined in the database.

- *sessionCatalog*

The *sessionCatalog* table saves the metadata of sessions. Each database record corresponds to a recorded session. The table has the following fields:

1. sessionID: a unique identification of a session created by the JVCR server.
2. sessionTitle: a title of the session.
3. smilFileURL: the URL of the SMIL document.
4. multicastAddress: the multicast address of the JVC audio/video RTP session.
5. videoPort: the UDP port used by JVC video streams.
6. audioPort: the UDP port used by JVC audio streams.
7. ttl: the ttl value of the RTP packet.
8. IPAddress: the IP address used by the JETS server.
9. whiteboardPort: the IP port used by the JETS whiteboard.
10. video: a boolean value to indicate whether the video stream is recorded or not.
11. audio: a boolean value to indicate whether the audio stream is recorded or not.
12. whiteboard: a boolean value to indicate whether the audio stream is recorded or not.
13. duration: the duration of the session.
14. sessionNote: the description of the session.
15. recorderName: the scribe's name
16. recordDate.

The primary key of the *sessionCatalog* table is *sessionID.*

- *eventLog:*

The *eventLog* table records the media objects involved in an event. It has the following fields:

1. sessionID.
2. objID: an identification of a media object. It is a unique value within a session.
3. mediaType: the type of a media object, such as "Text", "Image", etc.
4. chattext: the chat text of the JETS whiteboard.
5. CNAME: the CNAME of the JVC audio or video stream.
6. eventType: the type of event.

The primary key is constructed as *sessionID* + *objID*. Each record of *eventLog* links with a record of the *mediaObject* table through *sessionID* and *objID*.

- *mediaObject:*

The *mediaObject* table saves the attributes of each media object. It has the following fields:

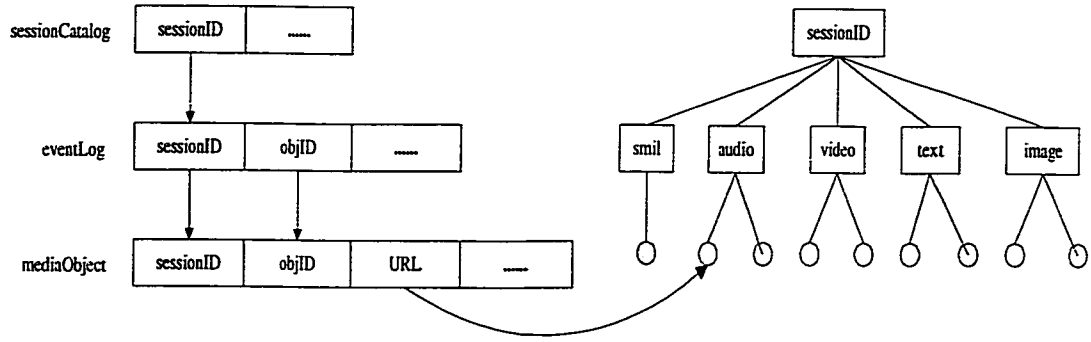
1. sessionID.
2. objID.
3. beginTimeStamp: the begin timestamp of a media object.
4. endTimeStamp: the end timestamp of a media object.
5. clipBegin: the beginning of a sub-clip of a continuous media object.
6. clipEnd: the end of a sub-clip of a continuous media object.
7. URL: the URL of the file which stores the media object.

The primary key is constructed as *sessionID* + *objID*.

Besides the database, there is a directory for each recorded session. The directory is named by the *sessionId* of the session. The directory has five subdirectories:

- SMIL subdirectory: stores the SMIL document created by the *smilCoder*.
- Video subdirectory: stores the captured video streams.
- Audio subdirectory: stores the captured audio streams.
- Text subdirectory: stores chat text.
- Image subdirectory: stores drawing annotations over whiteboard.

The relation between the database and the file system can be shown by the figure below [fig 5.16]:



Database Hierarchy file system

Figure 5.16 the relation between the JVCR database and file system

5.3 Implementation

According to the JVCR architecture and design presented, we implemented a JVCR prototype with JDK1.2 and JMF2.0 at the MCRLab, University of Ottawa. The system platform of JVCR is the same as that of JVC.

JVCR is implemented by Object Oriented Programming. The main classes and interfaces will be introduced in the following section. We also show how to use JVCR in the last session.

5.3.1 Implementation of the JVCR server

5.3.1.1 Class diagram

The class diagram of the JVCR server can be shown as follows [fig 5.17]:

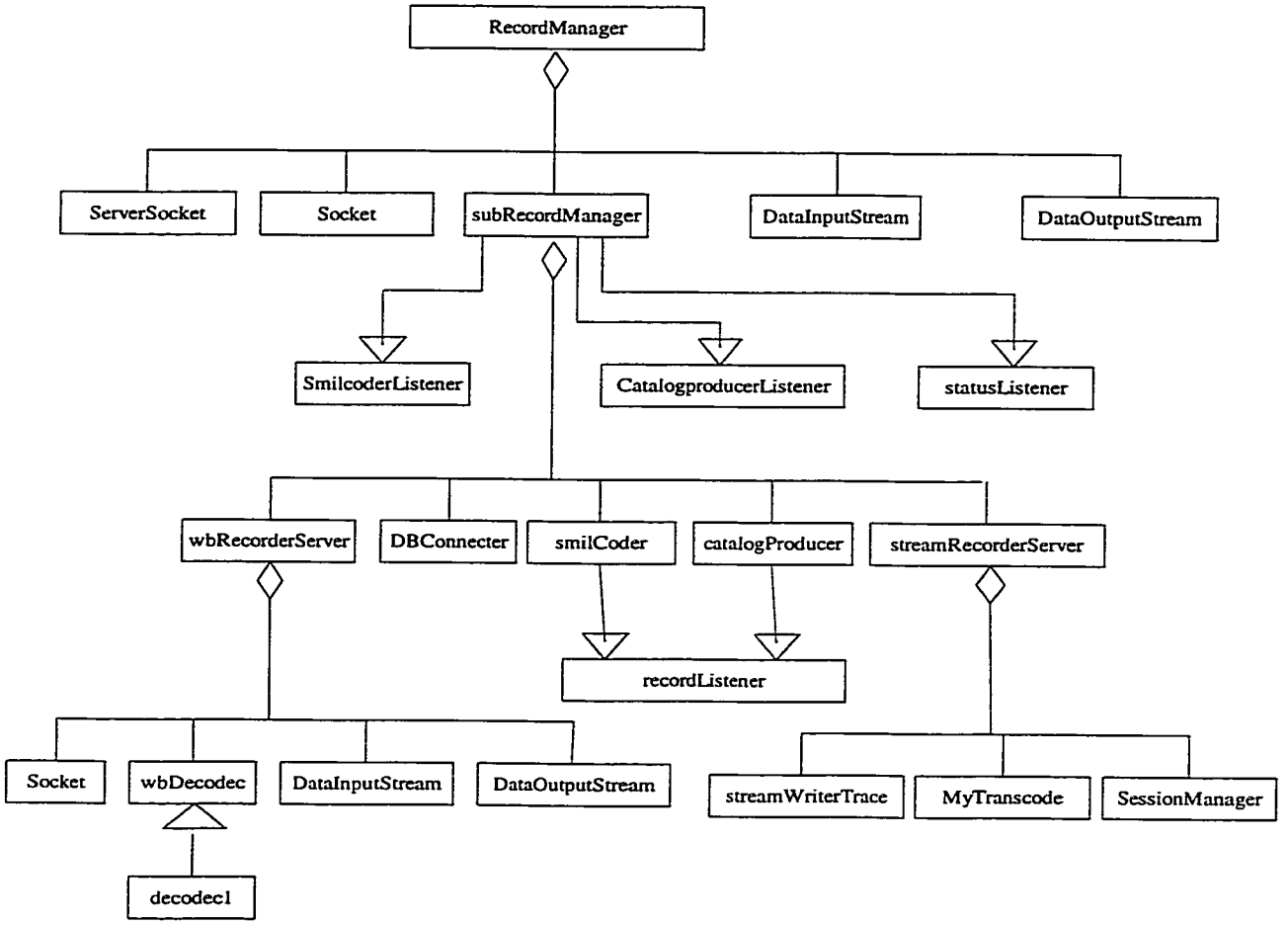


Figure 5.17 the class diagram of the JVCR server

5.3.1.2 Classes and Interfaces

Interfaces

The *RecordListener* defines methods to notify listeners about the client's interactive request.

The *SmilCoderListener* defines methods to notify listeners about the status of SMIL document creation.

The *CatalogProducerListener* defines methods to notify listeners about the status of the creation of the catalogue HTML page of recorded sessions.

The *StatusListener* defines methods to notify listeners about the initialization status of the recorder.

Classes

The *RecordManager* is implemented as a thread. Its *run ()* method creates the socket connected with the client, and spawns a new instance of the *subRecordManger.*

The *subRecordManager* is implemented as a thread. It implements the interfaces *SmilcodeListener, CatalogProducerListener* and *StatusListener.* The main task is done in *run ()* method, which includes an event handle loop.

The *wbRecorderServer* is implemented as a thread. It implements the *recordListener* interface. *wbRecorderServer* records the JETS whiteboard interaction. The dynamical finding and loading of a specific whiteboard event handler is done in *setDecodec(String decodecName)* method.

The *wbDecodec* is an abstract class. The class which implements event handler of the JETS whiteboard should extend *wbDecodec.* It defines two abstract methods: *translateData(DataInputStream wbin, long relativeTime, boolean recording)* handles events; *setParameters(Object para)* passes the parameters to *wbDecodec.* The parameter is a Java Object class, which is the super class of all Java classes. Therefore, the event handler can receive any type of parameter by casting it to the appopriate type in the specific application.

The *decodec1* extends *wbDecodec* and implements the event handler of the current implementation of the JETS whiteboard. It makes use of *JPEGImageEncoder,* a class of com.sun.image.codec.jpeg package, to save the annotations and encode the file in JPEG format.

The *streamRecorderServer* is implemented as a thread . It also implements *ReceiveStreamListener* and *RecordListener* interfaces.

The *MyTranscode* extends thread. It saves the RTP stream to the files.

The *smilCoder* is a thread to encode the SMIL document for the captured sessions.

The *catalogProducer* creates a catalogue page for recorded session in HTML format.

The *DBConnecter* is a helper class to access the database through the JDBC API.

5.3.2 Implementation of the JVCR client

5.3.2.1 Class diagram

The figure below is the class diagram of the JVCR client [fig 5.18]:

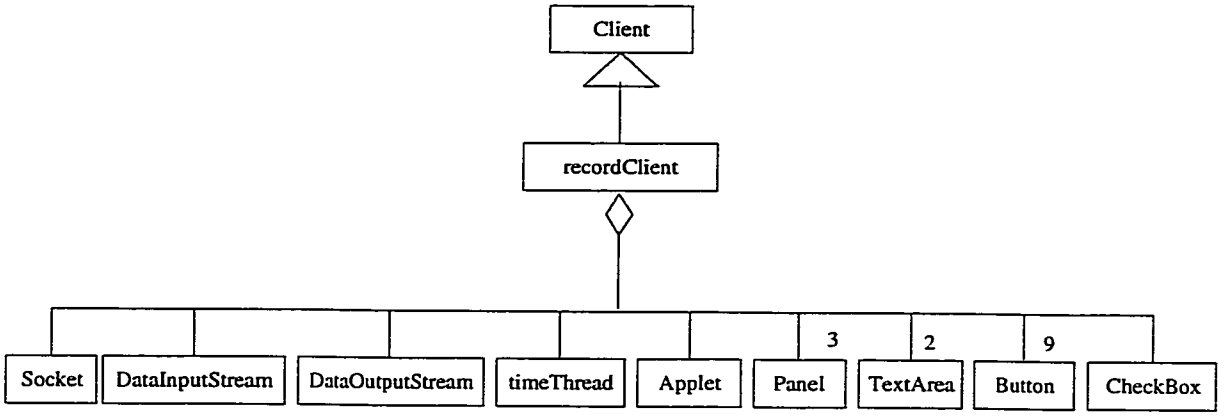


Figure 5.18 the class diagram of the JVCR client

5.3.2.2 Classes and Interfaces

The main classes of the JVCR client are *Client* and its subclass *recordClient.*

The *Client* extends the Java Frame class. It communicates with the JVCR Server through the DataInputStream and the DataOutputStream objects. Because the *Client* is a subclass of Frame, it is launched in an Applet *applet.* By passing the *applet* to *Client,* the *Client* can get the context and parameters of the Applet. This is required because the JETS whiteboard currently specifies the application-related parameters in the Applet parameter list instead of specifying them in a configuration file located in the JETS Server.

The *recordClient* extends the *Client.* The main task is to implement the user interface. It inherits all the public members and methods of *Client* as well as

implementing the *processData (DataInputStream in)* method, which receives the status of request execution from server.

5.3.3 Implementation of the JVCR data repository

Because the multimedia objects are stored in the file system, the JVCR database only needs to store textual data. Therefore, we implement the database *jetsevent.mdb* by the Microsoft ACCESS database. The ACCESS is a simple database that supports SQL. A datasource of *jetsevent.mdb* should be defined in Windows NT ODBC drivers so that the JVCR server can access it through the JDBC-ODBC bridge provided by JDK1.2.

5.3.4 Operation

5.3.4.1 Setup

Before using JVCR, we should configure the following programs:

- HTTP server

Configure the http server to reach the directory in which JETS has been installed.

- JMF2.0

Install the JMF2.0 in the same machine as the JVCR server.

- Web browser

Set the RealPlayer as a helper application of the browser.

- JVCR

Configure the JVCR client by specifying the parameters in the HTML page:

```
<APPLET CODE = jets.client.JetsStream CODEBASE = ./../.. WIDTH = 350 HEIGHT = 100 >
<PARAM NAME = tcpport VALUE ="7777">
<PARAM NAME = video VALUE ="On">
<PARAM NAME = session VALUE ="239.0.0.1">
<PARAM NAME = videoport VALUE ="9000">
<PARAM NAME = audio VALUE ="On">
<PARAM NAME = audioport VALUE ="9002">
<PARAM NAME = ttl VALUE ="1">
<PARAM NAME = dir VALUE ="i:\lding\jets2000\jets\media">
<PARAM NAME = chairperson VALUE ="true">
<PARAM NAME = recport value="6666">
```

<PARAM NAME = wbServerPort value="6793">
</APPLET>

The parameters are listed as follows:

tcpport: the TCP/IP port of recording server
dir: directory of media files
chairperson: load chair management function or not
session: the multicast address of Audio/Video tool.
videoport: the UDP port of video
video: transmit video or not
audioport: the UDP port of audio
audio: transmit audio or not
ttl: the time to live field of datagram delivery.
recport: record server port
wbServerPort: whiteboard server port

5.3.3.2 Start the server

Type the following line in the dos window to start the JVCR server:

java RecordManager tcp_port_num

tcp_port_num specifies the port that the JVCR server will listen to.

5.3.4.3 Start the client

The user should input the URL of the *jets2000.html* in the Web browser. He also should input his user name to log on. Then he can make a choice to record or play back by pressing the "record" button or the "playback" button.

5.3.4.4 Recording

First, the user inputs the session title and description and selects which streams are to be recorded. The recording can be divided into the following phases:

- Connect: click "connect" to initialize the record server. The JVCR server will sends back the prompt message when recorders are ready to record the streams.
- Start: click "start" button to start the recording process. The recorders start to record the specified streams. The timer begins to count the recording time.
- Stop: click "stop" button to stop the recording process.

The user can pause or resume the recording of a session:

- Pause: click "pause" button to pause the recording process. The recorders will ignore the received streams.
- Resume: click "resume" button to resume the recording process. The specified streams will be recorded again.

When the user finishes the recording of a session, he can create the SMIL document for this session and update the catalogue page.

- SMIL: click "smil" button to create the SMIL document. JVCR server searches all recorded sessions by this user and sends back the sessions that have not been created as SMIL documents yet. The user selects a session from the list, and JVCR client will send it to the JVCR server. The server then creates the SMIL document for this session.
- Catalog: click " catalog" button to create the index page for all recorded sessions.

To exit the JVCR client, the user just needs to click the "exit" button.

The following figure is a screen shot of the recording of both JVC and JETS whiteboard [fig 5.19].

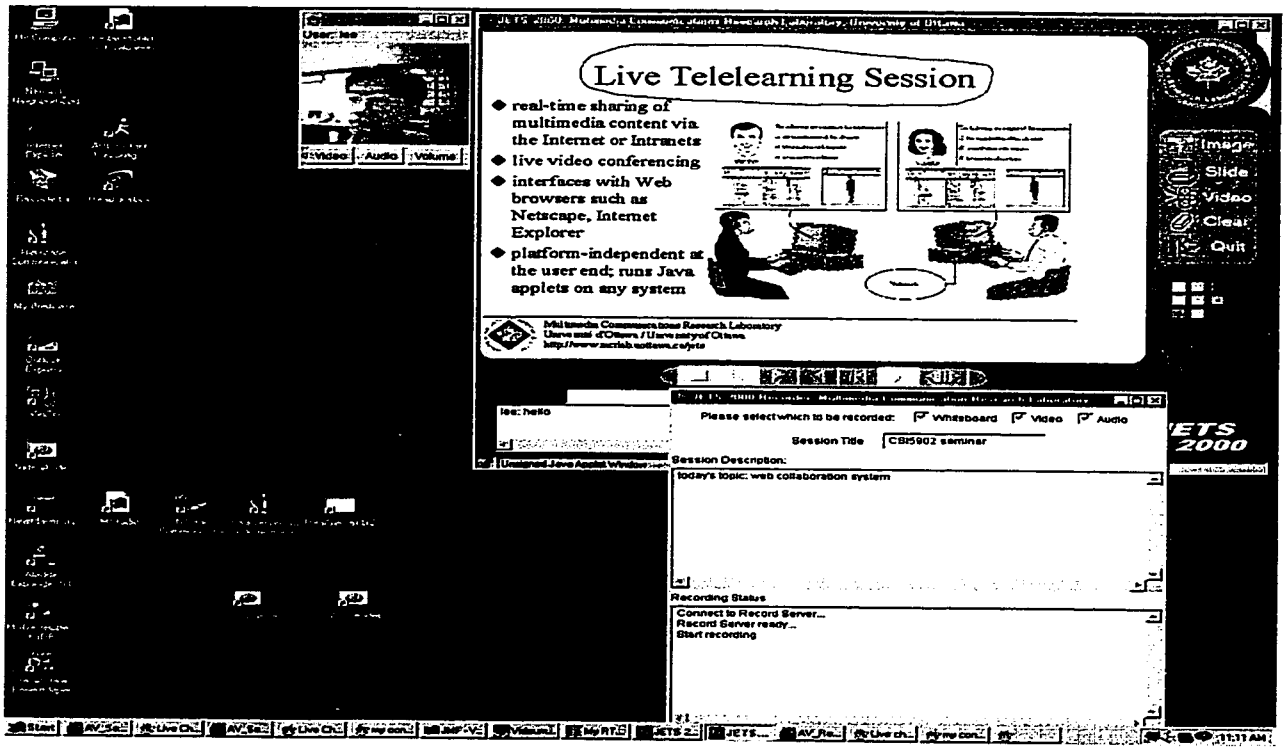


Figure 5.19 records the JVC audio / video streams and the JETS whiteboard

5.3.4.5 Playback

The user clicks the "playback" button in the *jets2000.html* page, then the catalogue of all recorded sessions will be listed in a browser window, shown in the following figure.

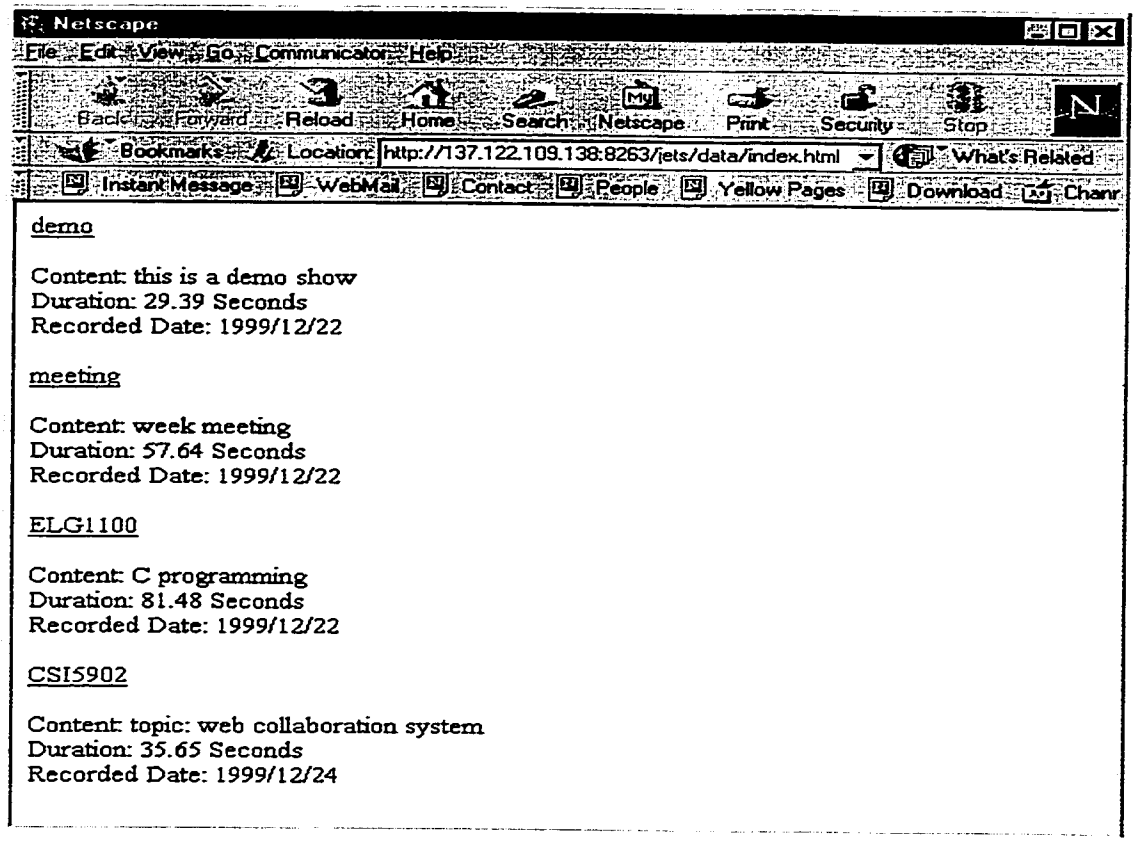


Figure 5.20 the catalogue of recorded sessions

To play back the session, the user just needs to click the link of that session, for example, "CSI5902". Then the Web browser will spawn a new RealPlayer window. The recorded session begins to play back in this RealPlayer window. The user can use RealPlayer to control the playback. The following screen shot shows the playback of a session in the RealPlayer.

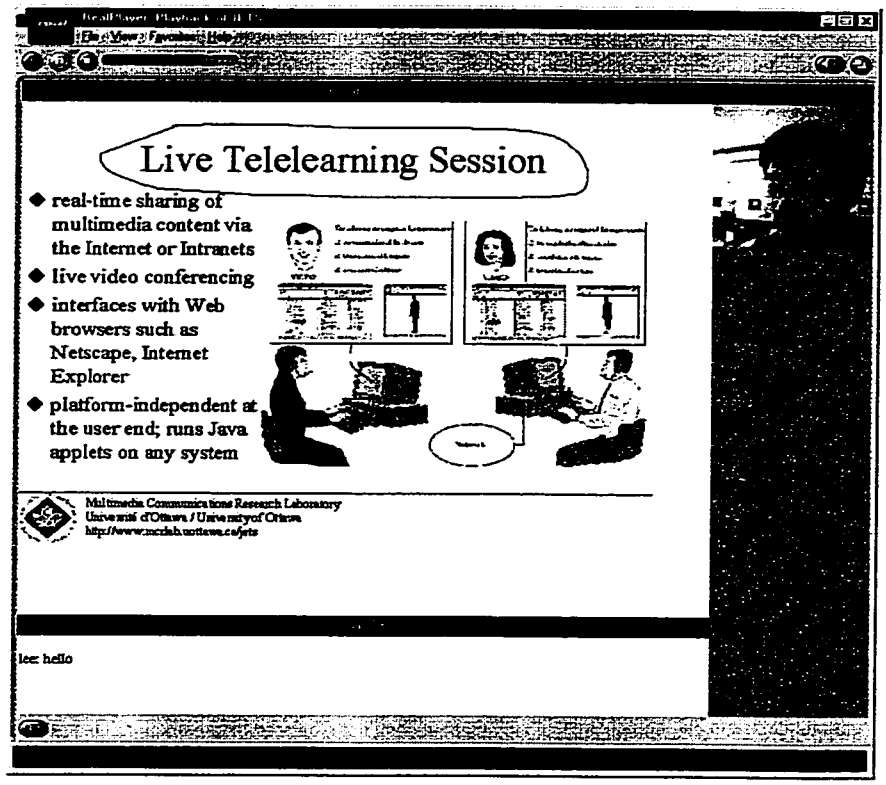


Figure 5.21 play-back of a recorded session

Chapter 6 Experiments

This chapter presents the experimental results achieved with the JVC and JVCR. We also describe the problems faced during the testing phase.

6.1 Environment

The JVC and JVCR, described in Chapter 4 and Chapter 5, have been tested in the MCRLab. The setup of our test environment is the following:

System hardware platform:

Four Pentium 333 PCs with 128M memory, audio and video capture devices. The PCs are connected to a 10 M Ethernet local network.

System Software platform:

1. Windows NT 4.0.
2. Various JDK releases, such as JDK1.1.7, 1.1.8, 1.2.1, 1.2.2.
3. Various JMF 2.0 releases, such as JMF 2.0 Early Access, 2.0 Beta, 2.0 Final Release. Each version of JMF has three subversions available:

- Pure Java: includes binaries written entirely in Java that can be installed on any operating system supported by the Java platform.
- Solaris Performance Pack: an optimized version for the Solaris platform that includes binaries for this operating system.
- Windows Performance Pack: an optimized version for the Windows platform that includes binaries for this operating system.

  We use the Windows Performance Pack for our testing.

4. Netscape Navigator and Internet Explorer.
5. RealPlayer G2 as SMIL player.

6.2 Test Results

We tested JVC and JVCR in the context of telelearning, a popular telecollaboration application domain.

6.2.1 JVC test results

We use the following test scenario:
The collaboration group has one instructor and three students. We capture one live audio stream and one live video stream of the instructor, and distribute them among the students. The students only receive the instructor's audio and video. They do not send out their live audio and video.

JVC was tested in all its functionality. As a test result, the audio and video can be captured and multicasted to the students. The students can receive and present the instructor's live audio and video. The video frame rate is 12-15 frames / second. The quality of video is very good. However, the audio has a long end-to-end delay. The quality of audio is acceptable.

6.2.2 JVCR test results

JVCR was tested in all its functionality. We used the following three test scenarios:
1. record JETS whiteboard
JVCR records all interactions within the JETS whiteboard. The instructor and the students freely interact with the whiteboard. For example, they can load to the shared workspace slides, images, H.263 video clips and make annotations. They can also do text chat with each other.

2. record JVC live audio and video
JVCR records the live audio stream and the live video stream of the instructor transmitted by JVC.

3. record JETS whiteboard and JVC live audio and video
JVCR records both the JETS whiteboard interactions and the JVC live audio and video. This scenario combines the above two scenarios.

As the test's result, JVCR successfully recorded all the collaboration information as well as audio and video streams in a session. The generated SMIL document is well – formatted and can be replayed by the RealPlayer correctly.

6.3 Limitations

At this point, a lot of implementation problems/bugs arise with JMF and SMIL because JMF and SMIL players are in their early evolution stages. This also limits the JVC and JVCR functionality and makes them more of a research prototype than industry-level products. However, these limitations are due to the immaturity of JMF and SMIL and not our architecture.

Currently, JMF2.0 is not stable enough and has a number of bugs. One major problem is that the JMF technology consumes a lot of memory and other resources. As an example, the CPU usage will jump to 100% if the JMF session is open for more than two live streams. Therefore, in our prototype we only permit one audio and one video stream to be captured and transmitted: those of the instructor of a session. Another problem is that JMF sometimes shuts itself down after about 10 minutes. Finally, JMF2.0 has a long capture device latency and a long end-to-end audio delay. These problems severely limit the usability of our prototype at the present moment.

As mentioned before, JVCR creates SMIL document according to SMIL1.0 specification. However, the current available SMIL players don't have a consistent behavior in supporting the SMIL specification. Some players don't support all feature of the SMIL1.0 specification, and some others have their own proprietary built-in extensions which can get in the way. Currently, we create SMIL document that can be played back by RealPlayer because this player is the most widely-used SMIL player.

For the RTP video and audio streams, users can select any available file formats supported by JMF to save them. Because the video/audio data are very huge, the disk will be filled very quickly, if we save them as uncompressed data. As of the writing time of this thesis, JMF2.0 can save a RTP stream as a file with compressed format such as MPEG or H.263. However, the RealPlayer doesn't recognize them yet and it will not play them. Therefore, we save RTP streams in uncompressed format at this stage.

6.4 Summary

The design and implementation of JVC and JVCR have successfully been tested with different versions of JDK and JMF. JVC can capture, transmit, receive and present audio and video streams among participants through RTP. JVCR successfully records all collaboration information that occurs on the JETS whiteboard and associated JVC live audio and video. The recorded session can be replayed by RealPlayer, one of the most popular SMIL players.

The experiments show many deficiencies of current JMF and SMIL player implementations. These deficiencies also limit the functionality of our prototypes. However, these limitations are due to the immaturity of JMF and SMIL and not to our architecture. Like any other technology, it is expected that all of the above deficiencies will be overcome in the near future as JMF and support for the SMIL format mature.

Chapter 7 Conclusion and Future Work

We conclude by discussing the design and implementation of JVC and JVCR, and suggesting directions for further investigation.

7.1 Conclusion

With the astonishing growth of the Internet and WWW, many Web collaboration systems have been developed. We believe that real time audio/video communication and recording of a collaboration session will greatly enhance the collaboration among the participants. These two advanced features are lacking in many systems or are provided by a platform- and application- dependent way in a few systems. In this thesis, we present the design and implementation of two portable collaboration tools to provide such features, i.e., the real time audio and video tool JVC and the recording tool JVCR.

JVC is a desktop multiparty audio and video tool over the Internet with client-server architecture. JVC has full functionality to capture, transmit and render the real-time audio and video of participants. It also provides a floor control mechanism to coordinate the usage of audio and video channels among the participants. Although the current prototype uses a simple "first come; first served" approach, JVC is flexible enough to apply a more complicated floor control algorithm which may be implemented in the future. JVC adopts a hybrid architecture that uses a centralized approach for state management and a distributed approach for the media data distribution. It uses reliable communication for control data between the server and the clients through TCP/IP sockets. All control data go to the server first and the server distributes them to all clients. On the contrary, the media data are fully distributed among all clients using RTP and IP multicast for the efficient transmission.

Portability is a key issue for the success of a collaboration system because it usually runs over different platforms. Currently, videoconferencing tools are platform dependent. A novel feature of JVC is its portability. Written with Java, JVC can run over any platform that is JMF-enabled. Furthermore, although the current JVC client is a stand-alone Java application, it can be easily transformed to an applet with the same functionality.

Finally, JVC is an open system. Based on JMF, JVC supports many standard audio and video codecs, such as H.263, H.261, G.711, JPEG, GSM, ADPSM etc. The media data can be wrapped into a wide range of file formats, such as QuickTime, AVI, AU, WAV, MPEG etc. JVC can also be integrated with other collaborative environments. This has been verified by linking JVC with the JETS collaborative environment.

Asynchronous collaboration is another advanced feature that is lacking in many Web collaboration systems. In this thesis, we also present the design and implementation of a recording tool, JVCR, which enables asynchronous collaboration. JVCR can capture real-time audio and video streams generated by JVC as well as all user interactions of JETS whiteboard. JVCR uses an approach that has not appeared anywhere else, thus far. First of all, all media components are recorded with standard formats, allowing their retrieval with the commonly available applications. Therefore, it offers the standard content that can be easily transformed or modified. A novel feature of JVCR is that it uses SMIL to specify the complex synchronization relationship among all media components. The user can play back the recorded sessions with any SMIL player. Therefore, the recording is decoupled from the playback. Furthermore, the recorded sessions can be easily assessed through the Web. By this way, asynchronous collaboration can occur anywhere and anytime.

Another advantage of the JVCR is its flexibility. JVCR is transparent to recorded applications and does not require changing the recorded application, so JVCR can be easily integrated with other collaboration tools. Furthermore, JVCR uses three-tier

architecture to access the backend database that is used to log the user interactions as well as the real-time media objects. The client GUI implements the presentation logic while the server application implements the data access logic. The database is accessed through standard JDBC interface. Although our current prototype uses Microsoft Access as a simple database, it can be replaced with a more powerful and higher performance JDBC compliant database without changing the JVCR source code.

Finally, JVCR is a portable system. It is implemented by pure Java technology and can be run on any platform that supports JVM. In particular, we implement the recording of the real-time audio and video streams based on JMF. Currently, we have not found other recording systems using similar technology.

7.2 Future work

Our work can be extended in several directions:

- Quality of Service management

The emphasis of our current work is to design and implement an end-to-end audio-video communication tool. Because JVC and JVCR need to transmit and process large amounts of audio and video data, effective support of end to end Quality of Service (QoS) has became increasingly important. In the future, we will investigate how to implement application level QoS management on JVC and JVCR. Some future work includes: specifying a set of user QoS parameters, such as resolution of video and quality of audio; mapping user QoS parameters onto QoS parameters for system components; monitoring QoS feedback from the RTCP packets; developing a QoS adaptation mechanism to deal with temporary changes in the available QoS parameters; and using RSVP to reserve resources.

- Enhancement of SMIL document

Currently, JVCR creates the SMIL document according to SMIL1.0 specification. SMIL1.0 is intentionally basic so that its concepts can be readily understood and easily implemented. However, this also limits the presentation ability of JVCR, for example, SMIL1.0 is weak in specifying the spatial layout. SMIL Boston, a new SMIL version, was released in August 1999. SMIL Boston adds new facilities for animation, extended navigation, and for handling multimedia delivered with broadcast audio and video. We will examine how to use these new features to enhance our system.

- Real-Time Streaming Protocol (RTSP) support

RTSP [38] is an application-level protocol that provides an extensible framework to enable controlled, on-demand delivery of real-time audio and video. Currently, we use RTP to transmit real-time data, but control of the RTP data is implemented by ad-hoc approach. Supporting RTSP, JVC and JVCR can provide a complete set of control functionality and be more flexible. For example, we can investigate how to control the JVCR recording process using RTSP's RECORD method. This work will still be based on JMF which will support RTSP in a future version.

Appendix A The SMIL document of a recorded session

The following is a sample SMIL document of a recorded session created by JVCR.

```
<smil>
 <head>
  <meta name="title" content="Playback of JETS"/>
  <meta name="copyright" content="Multimedia Communication Research Lab of University of
Ottawa"/>
  <layout>
   <root-layout height="640" width="800" background-color="white"/>
    <region id="wbtitleRegion" left="0" top="0" height="20" width="650" background-
color="navy"/>
    <region id="whiteboardRegion" left="0" top="20" height="520" width="650" background-
color="white" fit="scroll"/>
    <region id="chattitleRegion" left="0" top="540" height="20" width="650" background-
color="navy"/>
    <region id="textRegion" left="0" top="560" height="80" width="650" background-color="white"
fit="scroll"/>
    <region id="streamtitleRegion" left="650" top="0" height="20" width="150" background-
color="navy"/>
    <region id="livevideoRegion" left="650" top="20" height="620" width="150" background-
color="gray" fit="scroll"/>
    <region id="liveaudioRegion" background-color="gray"/>
  </layout>
 </head>
 <body>
  <par>
   <text id="title1" src="http://137.122.109.138:8263/jets\data\19991224111744\media\text\wbtitle.rt"
region="wbtitleRegion" fill="freeze"/>
   <text id="title2"
src="http://137.122.109.138:8263/jets\data\19991224111744\media\text\chattitle.rt"
region="chattitleRegion" fill="freeze"/>
   <text id="title3"
src="http://137.122.109.138:8263/jets\data\19991224111744\media\text\streamtitle.rt"
region="streamtitleRegion" fill="freeze"/>
   <img id="img1" src="http://137.122.109.138:8263/jets/media/images/mcrlab.gif"
region="whiteboardRegion" begin="00:00:00" end="00:00:03" fill="remove"/>
   <video id="video0"
src="http://137.122.109.138:8263/jets/data\19991224111744\media\video\3700.avi"
region="livevideoRegion" begin="00:00:00" end="00:00:35" clip-begin="npt=00:00:00"
fill="remove"/>
   <audio id="audio0"
src="http://137.122.109.138:8263/jets/data\19991224111744\media\audio\9496.au"
region="liveaudioRegion" begin="00:00:00" end="00:00:35" clip-begin="npt=00:00:00"
fill="remove"/>
   <img id="img2" src="http://137.122.109.138:8263/jets/media/slides/jets/img000.gif"
region="whiteboardRegion" begin="00:00:03" end="00:00:07" fill="remove"/>
   <img id="img3" src="http://137.122.109.138:8263/jets/media/slides/jets/img001.gif"
region="whiteboardRegion" begin="00:00:07" end="00:00:14" fill="remove"/>
   <img id="img4"
src="http://137.122.109.138:8263/jets\data\19991224111744\media\images\28340.jpg"
region="whiteboardRegion" begin="00:00:14" end="00:00:35" fill="remove"/>
   <text id="text1"
src="http://137.122.109.138:8263/jets\data\19991224111744\media\text\19991224111744_T1.rt"
region="textRegion" begin="00:00:00" end="00:00:35"/>
  </par>
 </body>
</smil>
```

Appendix B The catalogue of recorded sessions

The following is the sample source code of a catalogue HTML file for the recorded session.

```
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
  <META NAME="GENERATOR" CONTENT="Mozilla/4.04 [en] (WinNT; U) [Netscape]">
</HEAD>
<BODY>
<a href=http://137.122.109.138:8263/jets\data\19991222111144\smil\6292.smi>
demo<br>
</a>
<p>
Content:      this is a demo show<br>
Duration:     29.39 Seconds<br>
Recorded Date: 1999/12/22<br>
</p>
<a href=http://137.122.109.138:8263/jets\data\19991222111721\smil\4316.smi>
meeting<br>
</a>
<p>
Content:      C programming<br>
Duration:     81.48 Seconds<br>
Recorded Date: 1999/12/22<br>
</p>
</BODY>
</HTML>
```

References

1. S. Shirmohammadi, J.C. Oliveira and N.D. Georganas, *"Applet-Based Telecollaboration: A Network-centric Approach"*, IEEE Multimedia, pp.64-73, April/June 1998.

2. S. Shirmohammadi, *"Design and Implementation of JETS: A Java-Enabled Telecollaboration System"*, MASc. Thesis, University of Ottawa, 1996.

3. JETS API document, http://www.mcrlab.uottawa.ca/jets/doc/.

4. H. Eriksson, *"MBONE: The Multicast Backbone"*, Communications of the ACM, pp. 54-60, Vol.37, No.8, August 1994.

5. H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, *" RTP: A Transport Protocol for Real-Time Applications"*, IETF RFC1889, 1996.

6. W3C SMIL WebSite: http://www.w3.org/AudioVideo/#SMIL.

7. Real Network WebSite: http://www.real.com/.

8. JMF document, http://www.javasoft.com/products/java-media/jmf/index.html.

9. M. Handley, J. Crowcroft, C. Bormann and J. Ott, *"Very large conferences on the Internet: the Internet multimedia conferencing architecture"*, Computer Networks , pp.191-204, Volume 31, 1999.

10. T. Bray, J. Paoli, C.M. Sperberg-McQueen, editors, *"Extensible Markup Language (XML) 1.0"*, February 1998. Available at http://www.w3.org/xml/.

11. I.C. Chang, B.S. Liou, J.H. Huang et al, *"A Multimedia World Wide Web Based Conference Minute System for Group Collaboration"* , Multimedia Tools and Applications, pp.199-226, Vol.9, No.3, November 1999.

12. I.C. Chang, J.H. Huang, W.H. Tseng, *" Design and Implementation of a Multimedia CSCW platform"*, Multimedia Tools and Applications, pp.85-109, Vol.2, No.2, March 1996.

13. N.R. Manohar and A. Prakash, *"The Session Capture and Replay Paradigm for Asynchronous Collaboration"*, in H.Marmolin, Y.Sundblad, K.Schmidt (eds): Proceeding of the 4th European Conference on Computer-Supported Cooperative Work, ECSCW'95, Kluwer Academic Publishers, pp.149-164, 1995.

14. A. Schuett, R. Kata, S. McCanne, *"A Distributed Recording System for High Quality MBONE Archives"*, Proceedings of the First International Workshop on Networked Group Communication, (NGC '99), Pisa, Italy, November 1999.

15 S. McCanne, V. Jacobsen, *"vic: a flexible framework for packet video"* , Proceeding of ACM Multimedia 95, ACM, pp.511-522, October 1995.

16. V. Hardman, A. Sasse, M. Handley, et al, *"Reliable Audio for Use over the Internet"*, Proceeding of Inet'95, Honolulu, Hawaii, June 1995.

17. T. Berners-Lee, R. Fielding, L. Masinter, *"Uniform Resource Identifiers (URI): Generic Syntax"*, IETF RFC2396, August 1998.

18. J.S. Park, S.H. Lee, S.C. Kim et al, *"A Conferencing System for Real-Time, Multiparty, Multimedia Services"*, IEEE Transactions on Consumer Electronics, Vol. 44, No.3, pp.857-865, August 1998.

19. W. Holfelder, *"MBONE VCR: Video Conference Recording on the MBONE"*, Proceeding of ACM Multimedia '95, pp. 237-238, 1995.

20. W. Holfelder, *"Interactive Remote Recording and Playback of Multicast Videoconferences"*, the 4th. International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS '97), 1997.

21. L. Scott, R. Steve, et al. *"A Confederation of Tools for Capturing and Accessing Collaborative Activity"*, Proceeding of ACM Multimedia '95, pp.523-534, 1995.

22. S.Mukhopadhyay, B.Smith, *"Passive Capture and Structuring of Lectures"*, Proceeding of ACM Multimedia '99, 1999.

23. R.W.Hall, A.G.Mathur, F.Jahanian, et al, *"Corona: A Communication Service for Scalable, Reliable Group Collaboration Systems"*, Proceeding of ACM Conference on Computer Supported Cooperative Work (CSCW 96), Boston, MA, November 1996.

24. TANGO web site: http://tango.npac.syr.edu/tango/.

25. E. Craighill, R. Lang, M. Fong , et al., *CECED: "A System for Informal Multimedia Collaboration"*, Proceeding of ACM Multimedia'93, pp.437-443, 1993.

26. C.A. Ellis et al: *" Groupware – Some Issues and Experiences"*, Communications of the ACM, Vol.34, No.1, pp.38-58, 1991.

27. H.P. Dommel et al. *"Floor Control for Multimedia Conferencing and Collaboration"*, ACM Journal on Multimedia System, Vol.5, No.1, pp.23-38, January 1997.

28 T. Crowley et al. *"MMConf: An infrastructure for building shared multimedia applications"*, Proceeding of CSCW'90, ACM, New York, pp.637-650, 1990.

29 F. Fluckiger, *" Understanding Networked Multimedia Applications and Technology"*, Prentice Hall, New York, 1995.

30. N.R. Manohar and A. Prakash, *"A Flexible Architecture for Integrating Heterogeneous Replayable Workspaces"*, Proceeding of Third IEEE Int'l Conference on Multimedia Computing and Systems, Hiroshima, Japan, pp. 274-278, June 1996.

31. Mash project web site: http://mash.cs.berkeley.edu/mash/software/archive-usage.html.

32. J. Hughes , M.A. Sasse, *"Internet Multimedia Conferencing - Results from the ReLaTe Project"*, Proceeding of the ICDE World Conference, Pennsylvania State University, June 1998.

33. D.D. Clark, D.L. Tennenhouse, *"Architectural Consideration for a New Generation of Protocols"*. Proceeding of SIGCOMM'90, Philadelphia, PA, September 1990.

34. VAT website: http://www-nrg.ee.lbl.gov/vat/.

35. V. Hardman, A. Sasse, and I. Kouvelas, *"Successful Multi-Party Audio Communication over the Internet"*, Communications of the ACM, Vol. 41(5), pp. 74-80.

36. S. Floyd, V. Jacobson, S. McCanne, et al. *"A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing"*, Proceeding of SIGCOMM'95, Boston, MA, September 1995.

37. M. Handley, V. Jacobson, *"SDP: Session Description Protocol"*, IETF RFC 2327, April 1998.

38. H. Schulzrinne, R. Lanphier, A. Rao. *"Real Time Streaming Protocol (RTSP)"*, IETF RFC2326, April 1998.

39. E. Gamma, R. Helm, R. Johnson, et al, *" Design Patterns: Elements of Resuable Object-Oriented Software"*, Addison-Wesley Publishing Company, 1994.

40. S. Deering, *"Host Extensions for IP Multicasting"*, IETF RFC 1112, August 1989.

41. NetMeeting website: http://www.microsoft.com/netmeeting/.

42. M. Fowler, K. Scott, *"UML Distilled"*, Addison-Wesley Publishing Company, 2000.