An Approach for Recording Multimedia Collaborative Sessions: Design and Implementation

Shervin Shirmohammadi, Li Ding, and Nicolas Georganas
Multimedia Communications Research Laboratory
School of Information Technology and Engineering
University of Ottawa, Ottawa, Canada
[shervin | lding | georgana]@mcrlab.uottawa.ca

Abstract

In this article we present the design and implementation of a videoconferencing session recorder multimedia system. Our system is a videoconferencing tool that has the capability of recording live multimedia collaborative/conferencing sessions. The main distinction between our architecture and other ones is that unlike other recording systems, which require the original application for the playback, our system generates Synchronized Multimedia Integration Language (SMIL) compliant documents which can be played back in any SMIL-compliant player.

1. Introduction

The ability to record a multimedia conferencing/collaboration session is an important requirement for many applications. In many cases, it is necessary to play back the events that took place in a meeting, in the exact same order as the events occurred in the live session. For example, when a student misses a live telelearning session of a virtual classroom, where instructors and students had met to discuss a specific lecture, the absent student can play back exactly what happened in the virtual classroom, including conversations and the documents that were presented and discussed among participants.

Therefore it becomes beneficial for multimedia conferencing/collaboration systems to provide some sort of a recording mechanism. As an analogy one can consider recording meetings in the real world; i.e., using video camcorder to record a wedding or birthday or any other meeting. The differnce in video-conference sessions is that the meeting is between geographically distributed participants.

In this paper we present the design and implementation issues that we came across when creating *J-VCR*: the *Java Video Conference Recorder*. J-VCR subscribes as a client to our JETS[1] multimedia collaboration system [1] and records the live whiteboard sessions together with real-time video and audio streams. It converts, on the fly, its recorded archive into SMIL-compliant files that can be played back in any SMIL [19] player.

Some parts of our design have understandably been specifically implemented for the JETS system, but the overall architecture and approach can be generalized for any similar conferencing/collaboration tool.

The next chapter of this paper discussed the fundamental design issues and

[1] JETS: Java-Enabled Telecollaboration System.

requirements, while chapter 3 presents the overall architecture. Chapter 4 consists of our implementation, and chapter 5 and 6 conclude the article with discussions about similar work and final remarks.

2. Design

In this section we discuss the functional requirements and design considerations of the system.

2.1 Design Goals

The overall goal of J-VCR is to enable effective asynchronous collaboration by recording and playback through providing a session archive service. In order to achieve this goal, some subgoals have been identified:

- *Unobstructive recording*

The J-VCR should be an "added-value" feature supporting the natural activities of the collaborative session. Unobstructive recording refers to transparent recording of the recorded applications. Generally, applications are "recording-unaware"; i.e, they are not designed with recording in mind. But we believe that one should not have to change the behavior of an application in order to be make it recording-aware. In other words, from the development point of view, no source code or other systems' components of the original application should be changed.

- *Integrated service*

A collaboration system may consist of applications with different architectures, service interfaces and media broadcast approaches. For our project, we need to record the output of two different components: JETS shared applets (telecollaboration) and J-VC[2] (videoconferencing), our real-time videoconferencing tool. JETS uses an replicated event-based 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). J-VC, on the contrary, uses RTP [6] over MBone [4] to transmit the audio and video streams. Every participant can receive the streams if he joins the multicast group. Therefore, J-VCR should provide an integrated service interface so that it can work with different collaboration applications and hide all these difference to the participant.

- *Open archive format*

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

- *Interactive recording*

J-VCR should provide an interactive control ability so that users can customize it to satisfy their specific requirements. To be more accessible, J-VCR should provide a platform-independent, intuitive and easy-to-use graphics user interface for the users to issue their recording request and control command.

[2] J-VC: Java Video Conference. This is the conferencing part of J-VCR.

2.2 Design Considerations

Let us have a more detailed look at some key issues that influence our design. Generally, a complete recording process includes the following stages:

- Capture: captures the collaboration content.
- Handle: processes the collaboration content.
- Store: saves the handled collaboration content to database so it can be used asynchronously.
- Playback: plays back the recorded content.

Capture

To capture the user interactions of JETS and the associated real-time audio/video streams, J-VCR joins the session as a *special* JETS participant. The difference from other participants is that J-VCR basically participates as a receiver only, and doesn't send out any media streams to be broadcasted while other participants may be a senders as well as a receivers at the same time [fig. 1].

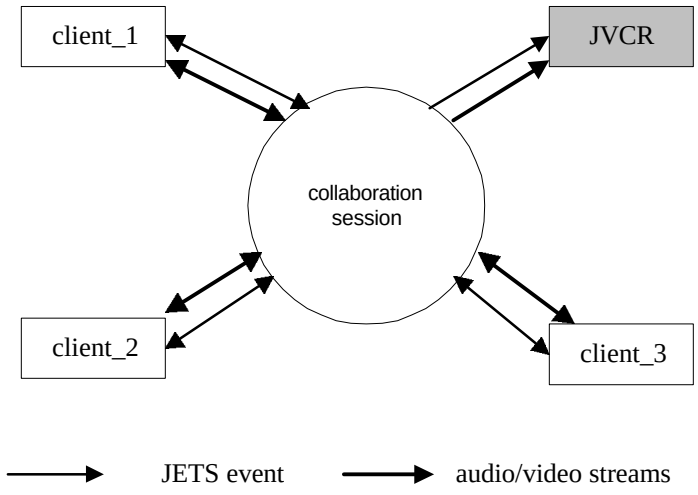


Figure 1 J-VCR joins the session as a participant

J-VCR uses two recorders, one for JETS applications and another for J-VC. 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 which decreases the workload of the server.

J-VCR makes use of the Java Media Framework (JMF) [17] to capture, handle and store RTP audio of video data. 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.

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.

Handle

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 [3].

- *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 need 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 J-VCR is to create application-independent archive data. Therefore, J-VCR 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 of 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 J-VCR 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, J-VCR needs an application dependent event handler which knows the semantics of events of specific implementation of the JETS whiteboard. Whenever J-VCR captures an event through the event receiver, it will delegate the event to event handler. The event handler will explain each event in order to identify the multimedia objects used in the event. 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.

A general process of generating a media object from a JETS event is given as follows [fig. 2].

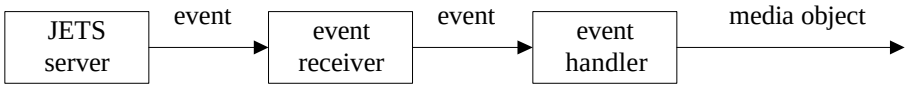


Figure 2 A general process to extract a media object

As we can see, the event handler does not connect to the JETS server directly and the event is passed from 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 J-VCR 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 event handler should be dynamically found and loaded by J-VCR through its name registered in the configuration file using Java's dynamic loading mechanism.

J-VCR need to preserve the temporal relationship among different objects. It 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. J-VCR is designed to work with recording-unaware applications, which means J-VCR 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, J-VCR will generate the timestamp for the received event. Because the recorder for JETS and recorder for J-VC 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, J-VCR 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 J-VC audio/video stream is received by the J-VCR 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, J-VCR receives event1, which loads *img1.jpg* into the whiteboard; and in time t2, J-VCR receives event2, which load another image to replace *img1.jpg*. Then, the *active time* of img1.jpg is t2-t1. [fig 3]

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 3 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.

J-VCR creates a SMIL document to combine all media objects together by specifying the synchronizaiton relationship among media objects. The Synchronized Multimedia Integrated Language (SMIL, pronounced as "smile") 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 [20] 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[18] .

Store

The captured media objects and associated temporal relationship should be saved in a repository for later playback. J-VCR 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.

In order to be flexible and not be bound to a specific kind of database, J-VCR accesses the database through the Java Database Connect (JDBC) API .

playback

Collaboration sessions are recorded as SMIL documents, so they can be played back by SMIL players. To facilitate access to the recorded sessions, J-VCR 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 J-VCR 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.

3. Architecture

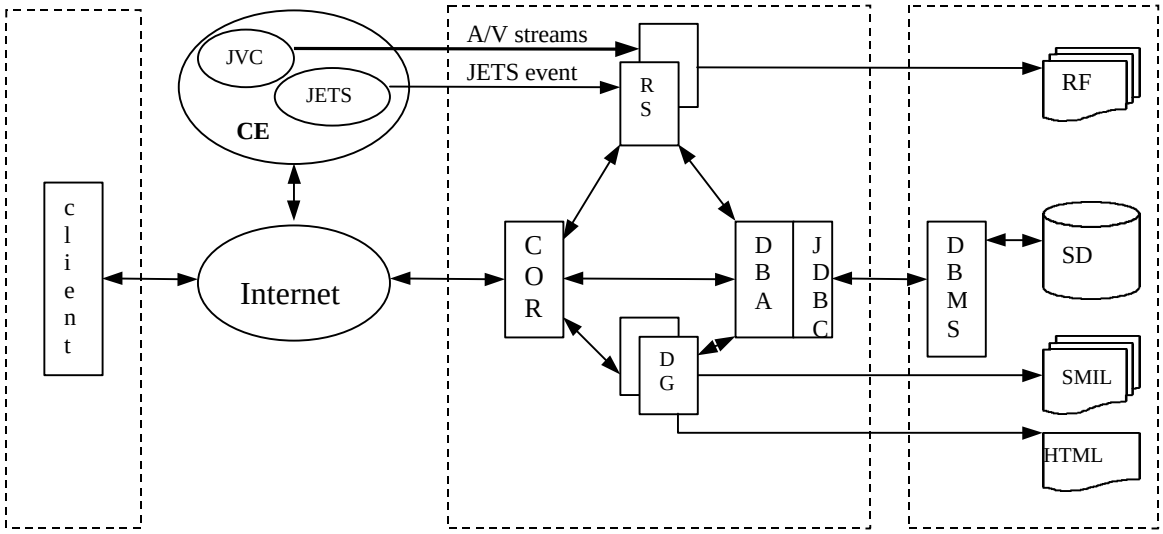
After presenting the requirements of the recording tool, we will now present in detail the component-level architecture of our system.

3.1 System Architecture

J-VCR is a client-server system with three parts:

- J-VCR client
- J-VCR server
- Data repository

The following figure shows the architecture of J-VCR.[fig 4]



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 4 system architecture of J-VCR

The *J-VCR Client* provides a graphics user interface to access J-VCR services. It is an applet and can be run in any Java-compliant Web browser.

The *J-VCR 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 J-VCR 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 which capture, handle and store the session data. RS consists of the JETS recorder *wbRecorderServer* and the J-VC recorder *streamRecordServer*. The JETS recorder connects with the JETS server through TCP/IP while the J-VC recorder joins the J-VC 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 Assessor): the database access interface between the J-VCR 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 J-VCR archive data. It include the following components:

- RF: resource files that 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: a SQL database which saves the links of media objects and their *active time*.
- SMIL documents: the documents which 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.

3.2 Design of the J-VCR server

In this section, we give more details of the J-VCR server design.

3.2.1 COR (Coordinator)

COR provides a unified service interface to clients. It is the coordinator component of the J-VCR 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 J-VCR 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.

subRecordManager

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

1. Receives all requests from J-VCR 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 / receiver 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 *streamRecordServer.*

The *subRecordManager* gets the client request as well as sends back the feedback to the client. In order to dispatch the client request to the appropriate objects as well as get the status of other J-VCR server components, *subRecordManager* should have a bi-directional communication with these objects. 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 *streamRecordServer*, the multicast address, audio and video RTP port, and TTL may be different for different recording clients. In both cases, the J-VCR client can specify these parameters and send them to the J-VCR 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.

3.2.2 RS (Recorders)

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

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 informed of the state change.

The *wbRecorderServer* works as a special JETS whiteboard client. It runs on the server side and saves the shared media object without rendering 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 J-VCR client. The file name can be gotten directly because it has already been included in the JETS event message. For example, the following event is to load an image to whiteboard (fig 5):

2	image001.gif

Figure 5 an example of a JETS event

For a dynamic media object, a file will be created first to store the 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.

streamRecordServer

The *streamRecordServer* records J-VC audio and video streams. The responsibilities of *streamRecordServer* include:

1. Joins the corresponding J-VC 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 *subRecordManager* state listener and does appropriate action when informed of the state change.

streamRecordServer makes use of JMF RTP API to receive and store the audio/video RTP streams.

In order to receive audio or video RTP streams, *streamRecordServer* implements the interface *ReceiveStreamListener* of JMF RTP API. It will get notification when a new stream has been received. Then it launches a new *RTP writer*, which is a thread to write the detected RTP stream to disk as a seperate file. *RTP writer* creates

JMF *Datasink* and *Processor* object from the stream and uses them to control the recording of the stream.

As the *wbRecorderServer,* the *streamRecordServer* 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*.

3.2.3 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.

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.

catalogProducer

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

3.2.4 DBA (DataBase Accesser)

The *DBA* is a thin wrapper of JDBC which hides JDBC API from the database service client as well as provides an unify 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.

3.3 Design of the J-VCR client

The J-VCR 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 J-VCR server.

Communication between J-VCR client and server goes through the TCP/IP socket. Once downloaded into the user machine, the client applet establishes a Socket connection to the J-VCR server. 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 J-VCR Server in the initialization phase of recording. The J-VCR server then uses these parameters to initialize the whiteboard and Audio / video recorders.

The message transmitted between J-VCR client and server is packaged as a packet with the following format [fig 6]:

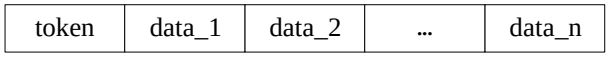


Figure 6 the packet format transmitted between the J-VCR client and server

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 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 can be enabled or disabled according to the current recording status to prevent misuse.

3.4 Design of the J-VCR data repository

The J-VCR 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.

- *SessionCatalog:* saves the metadata of sessions. Each database record corresponds to a recorded session.
- *eventLog:* saves the begin timestamp and end timestamp of each media object.
- *mediaObject:* saves the attributes of each media object.

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 7]:

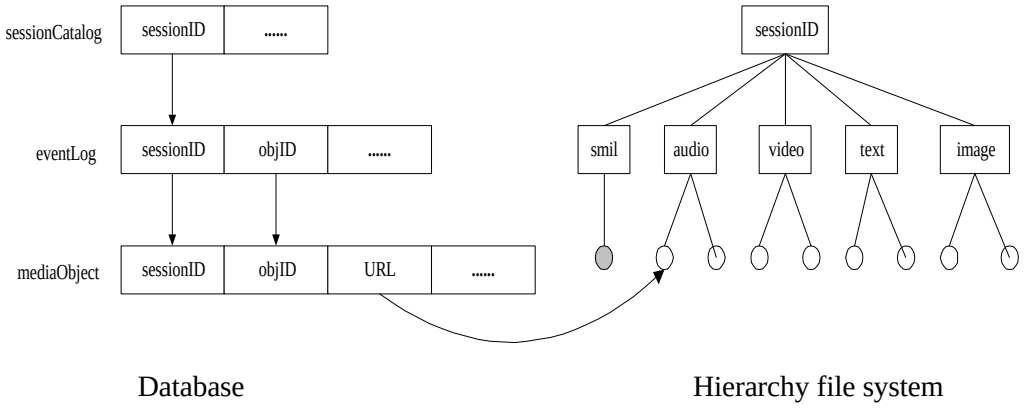


Figure 7 the relation between the J-VCR database and file system

4. Implementation

The architecture described in the previous section was implemented using JDK 1.2, JMF, and JDBC. The database itself was an ODBC database running on Microsoft Windows NT 4.0 Workstation. A sample JETS collaboration session was started and then coupled with the J-VC conferencing tool. Figure 8 shows a screen shot of the session.

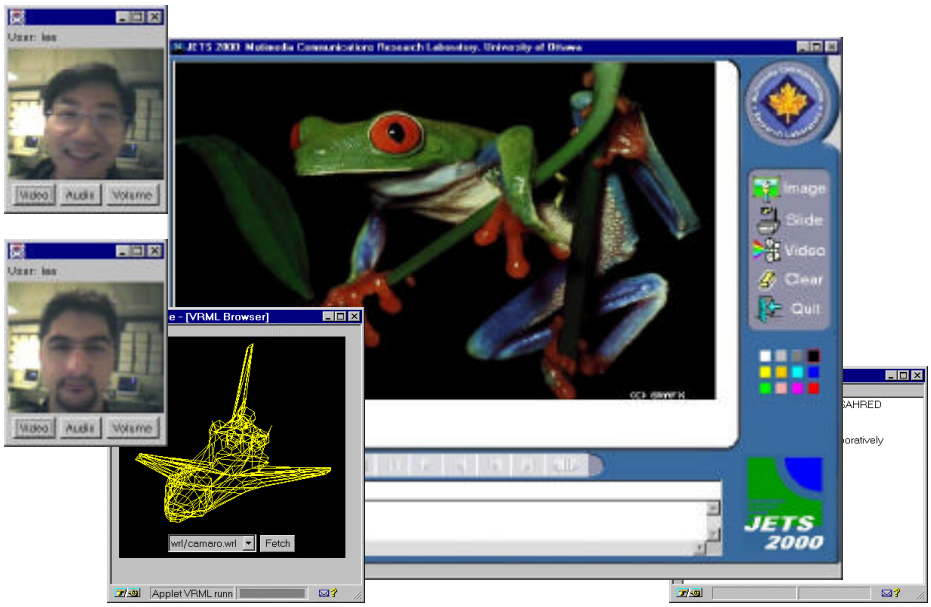


Figure 8. A JETS/J-VC collaboration session between two users.

In the following section, we discuss some of the implementation details, as well as problems that we encountered.

4.1 Implementation issues

The J-VCR recording Client is a Java applet to make it is possible to access the recording service from any Java-enabled Web browser. The J-VCR recording server is implemented as a multithread Java application in order to get high performance. WbRecordServer, StreamRecordServer, SmilCoder, and CatalogProducer are all implemented as a separate threads. StreamRecordServer is implemented based on the JMF API. We created two separate *SessionManager* objects of the JMF API for the video and audio streams, and we implemented the ReceiveStreamListener interface to receive the A/V data. A synchronization mechanism is applied in the callback method to coordinate shared resource access.

The J-VCR server accesses the back-end database through SQL language and JDBC API. For simplicity, we use Microsoft Access which provides the necessary ODBC drivers and can be accessed by JDBC through JDBC-ODBC bridge.

When creating the SMIL document, we use predefined layout template to setup the regions. There are two reasons: first, it is difficult to customize the layout as same as the original whiteboard because of the limitation of spatial layout specification of SMIL. Another reason is that we want to combine the interaction of whiteboard and video/audio into an integrated presentation on the same window. The Body part of the generated SMIL document consists of a “par” element, and all multimedia

components in eventLog as the children of that “par” element with explicit begin and end time of presentation. Figure 9 shows a recorded session being played back in RealNetwork's RealPlayer tool.

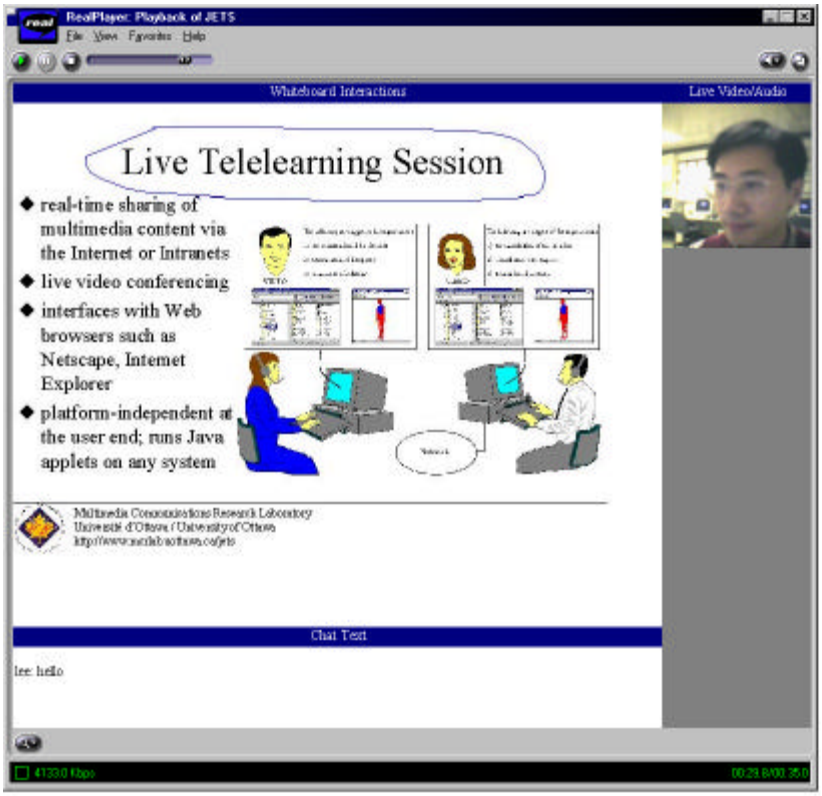


Figure 9. A recorded J-VCR session played back in RealPlayer.

As can be seen from the picture, all whiteboard, chat, video and audio components play back synchronously as one continuos session. Any SMIL player can playback the recorded sessions. Users just need to click the “playback” button, which will launch a new Web browser window that contains the index HTML page for all recorded sessions automatically. By setting SMIL player as a helper application of the Web browser, users just need to click on the appropriate link and the session will play back. The SMIL player will synchronize the presentation of every media according the SMIL document specification. The client can also use some advanced features provided by the player such as forward, backward, and pause to browse through the recorded session.

4.2 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 J-VCR functionality and makes it more of a research prototype as opposed to an industry-level product. 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 recorded: those of the instructor of a session. Another problem is that JMF sometimes shuts itself down after about 10 minutes. These problems severely limit the usability of our prototype at the present moment.
As mentioned before, J-VCR 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 support by JMF to save them. Because the video/audio data is very huge, the disk will be exhausted very shortly if we save them as uncompress data. As of the writing time of this article, JMF2.0 can save RTP stream as a file with compressed format such as MPEG or H.263. However, RealPlayer doesn’t recognize them yet and it will not play them. Therefore, we save RTP streams as uncompress format on this stage.
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.

5. Related Work

The recording system can be found in three application domains: videoconferencing system, collaboration system (without real-time videoconferencing support) and multimedia document authoring systems.
Most of the recording systems for videoconferencing are MBone tools that are used to record MBone videoconferencing sessions. MBone VCR [15] is the first application that supported interactive, synchronized recording and playback of MBone sessions. However, it was a single-user application that simply dumps packets to a local file and cannot be access through a distributed system. MBone VCRoD [16] is a client-server based system for interactive remote recording and playback of MBone sessions. Schuett et al also present a recording system for MBone conferencing system [2]. It uses multiple distributed recorders placed close to the source of session in order to get high quality archives.

For tightly coupled collaboration applications [8], there are relatively few recording systems that can be found in the literature. Manohar and Prakash present a system to capture and replay a session for asynchronous collaboration [9]. In this system, a session with an application’s user interface is encapsulated into a data artifact, referred to as "session object". Each session object is composed of several data streams that encapsulate audio annotations and user interactions with the application.

The replay of a session object is accomplished by dispatching these data streams to the application for re-execution. Another system is MINUTE [7]. It can record users' interaction with the shared applications and live audio and video streams form a WWW based conferencing system. The recorded session is composed of nodes, where each node can be played back independently. However, the nodes must be created manually. 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 J-VCR 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 which will play them back. By contrast, J-VCR 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 J-VCR 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. J-VCR, as we mentioned before, can record the streams transmitted by both RTP and TCP.

In addition to the above telecollaboration tools, there are some "presentation recording" systems that create multimedia documents dynamically and support playback later on [5][10][11][14]. But these systems focus on multimedia document creation: they record live activities in the real world, not the users' interactions with the computer support telecollaboration application. They are used in meeting rooms or classrooms with special equipment, which makes them much less accessible compared to our web based telecollaboration system.

6. Conclusions and Future Work

The architecture and design of the J-VCR recording system was presented in this article. J-VCR provides a unique solution for supporting real-time videoconferencing and recording of a telecollaboration session. Developed with Java, our approach provides a portable recording tool for JETS whiteboard as well as J-VC real-time audio/video tools. J-VCR creates and manages audio and video streams among participants, as well as creating reusable multimedia documents on the fly, which preserves the temporal relationship of live collaboration at no cost.

Initially, the emphasis of our work was to design and implement an end-to-end audio-video communication tool with recording capabilities. In the next phase, we will focus on QoS management. The QoS monitoring parameters can be obtained from RTCP packets. Another area of research is SMIL itself. SMIL1.0 has intentionally been kept basic so that its concepts can be readily understood and easily implemented. However, this also limits the presentation's ability, for example, SMIL1.0 is weak in specifying the spatial layout. Other works such as SMIL Boston, a new SMIL version, try to add new facilities for animation, extended navigation, and for handling multimedia delivered with broadcast audio and video. We will research how to use these new features in our system.

References

[1] S. Shirmohammadi, J.C. Oliveira and N.D. Georganas, "Applet-Based Telecollaboration: A Network-centric Approach", IEEE Multimedia, Volume 5, Number 2, April-June 1998, pp. 64-73.

[2] A. Schuett, R. Kata, S. McCanne, “A Distributed Recording System for High Quality MBone Archives”, Proc. of the First International Workshop on Networked Group Communication, (NGC '99), Pisa, Italy, November 1999.

[3] E.Craighill, R.Lang, M.Fong , and K.Skinner, “CECED: A System for Informal Multimedia Collaboration”, Proc. Of ACM Multimedia’93, pp437-443.

[4] H. Eriksson. 1994. "MBONE: The Multicast Backbone," Communications of the ACM , pp54-60, Volumn 37, Number 8, August 1994 .

[5] G. Cruz and R. Hill, “Capturing and Playing Multimedia Events with STREAMS”, Proc. Of ACM Multimedia’94.

[6] H.Schulzrinne, S.Casner, R.Frederick, V.Jacobson “RTP:A Transport Protocol for Real-time Applications”. IETF RFC 1889.

[7] I.C. Chang, B.S. Liou, J. H. Huang etc. “A Multimedia World Wide Web Based Conference Minute System for Group Collaboration “, Multimedia Tools and Applications, pp.199-226, Volume 9, Number 3, November 1999.

[8] 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.

[9] N. R.Manohar and A. Prakash, “The Session Capture and Replay Paradigm for Asynchronous Collaboration”, in H.Marmolin, Y.Sundblad, K.Schmidt9eds): Proceeding of the 4th European Conference on Computer-Supported Cooperative Work, ECSCW’95, Kluwer Academic Publishers, 1995, pp. 149-164.

[10] G.D. Abowd, C.G. Atkeson, J.A. Brotherton, etc. “Investigating the capture, integration and access problem of uniquitious computing in an educational setting”. Proc. of ACM CHI’98.

[11] S. L. Minneman, S. R. Harrison, B. Janssen, etc. “A Confederation of Tools for Capturing and Accessing Collaborative Activity”. Proc. ACM Multimedia ’95, pp.523-534.

[12] T. Berners-Lee, R. Fielding, L. Masinter, “Uniform Resource Identifiers (URI): Generic Syntax”, IETF RFC2396, Aug.1998.

[13] W.Hurst, R.Muller , “A synchronization Model for Recorded Presentations and its Relevance for Information Retrieval”, Proc. ACM Multimedia ’99.

[14] S. Mukhopadhyay and B. Smith, “Passive Capture and Structuring of Lectures”,

Proc. Of ACM Multimedia ’99.

[15] W. Holfelder , “MBone VCR: Video Conference Recording on the MBone”. Proc. of ACM Multimedia ’95, pp. 237-238.

[16] W. Holfelder: “Interactive Remote Recording and Playback of Multicast Videoconferences” 4th. International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS '97).

[17] JMF website: http://www.javasoft.com/products/java-media/jmf/index.html

[18] Real Network website: http://www.real.com/

[19] W3C SMIL website: **http://www.w3.org/AudioVideo/#SMIL**

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