Université d'Ottawa · University of Ottawa

Effective Tools for Transparent Synchronous Collaborative Environments

by

Dongsheng Yang, B.E.

A Master's thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master's of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

Abstract

Synchronous collaborative environments can provide an identical visual and operable working area among geographically separated participants. There are two basic approaches for providing a shared workspace. The first one, collaborative-aware approach, relies on the development of collaborative-aware applications that directly support multiple simultaneously active user inputs and synchronization of outputs to all participants. The second one, collaborative-unaware approach, introduces some intermediate layers between existing single-user applications and the underlying window management system. These layers enable single-user applications to be shared transparently. In order to reuse the large amount of existing single-user applications, most users choose the collaborative-unaware approach. However, to build collaborative-unaware systems to share single-user applications transparently is a great challenge.

This thesis describes the design and implementation of some transparent synchronous collaborative tools. They are: (1) the latecomer support for Java applications, Java applets and JMF players; (2) the client synchronization to minimize data transmission latency; (3) the light-weight multi-session support to let different collaboration groups work at the same time and (4) the collaborative VRML viewer to integrate the blaxxun3D VRML viewer with collaboration systems. These tools optimize existing transparent synchronous collaboration systems and make them more realistic, more complete and more generic.

Acknowledgements

My sincere gratitude goes to my academic supervisor Dr. Nicolas D. Georganas, who helped me, encouraged me, and guided me towards my academic as well as my personal success. The thesis would also not be successful without the help from my co-supervisor Dr. Abdulmotaleb El Saddik. He provided me a substantial amount of help and suggestions throughout my work. A special gratitude also goes to fellow students of the DISCOVER laboratory team for many helpful discussions and their constant encouragement and support.

I would like to give my sincere thanks to my husband and parents for their support through out my graduate studies. I am also grateful to all my friends and family who helped me either with suggestions or with positive affirmation.

The financial support of the National Capital Institute of Telecommunications (NCIT) given to the University of Ottawa under the Distributed Virtual Environments with Training Applications (DIVERTIONS) project is also gratefully acknowledged.

Table of Contents

List of Figures

List of Abbreviations

API: Application Programming Interface

AWT: Abstract Window Toolkit

CBE: Collaboratory Builder's Environment

CHIME: Columbia Hypermedia IMmersion Environment

DG III ACTS: Directorate-General III

DISCIPLE: DIstributed System for Collaborative Information Processing and Learning

DVE: Distributed Virtual Environment

EAI: External Authoring Interface

GIF: Graphics Interchange Format

GUI: Graphical User Interface

HTML: Hyper Text Markup Language

I/O: Input/Output

IP: Internet Protocol

ITU-T: International Telecommunication Union - Telecommunication

JASMINE: Java Application Sharing in Multi-user INteractive Environment

JCE: Java Collaborative Environment

JETS: Java Enabled Telecollaboration System

JMF: Java Media Framework

LEVERAGE: LEarn from Video Extensive Real Atm Gigabit Experiment

P2P: Peer-to-Peer

SCM: Session Control Manager

SRM: Scalable Reliable Multicast

SWS: Shared Window System

TCP: Transmission Control Protocol

TTL: Time-To-Live

UDP: User Datagram Protocol

UI: User Interface

VRML: Virtual Reality Modeling Language

WMS: Window Management System

Chapter 1 Introduction

1.1 Thesis Background and Motivation

Collaborative environments through networks have become a very popular research area for many years. The growth of the Internet makes the world smaller and more and more people would like to work together from places that are geographically distributed. What people need is a shared workspace similar to physical meeting rooms in the real world. The shared workspace can provide an identical visual and operable working area among geographically separated participants. It is one of the most important features of synchronous collaboration systems. There are two basic approaches to provide a shared workspace. The first one is called collaborative-aware approach. It relies on the development of collaborative-aware applications that directly support multiple simultaneously active user inputs and synchronization of outputs to all participants. The second one is called collaborative-unaware approach. It introduces some intermediate layers between existing single-user applications and the underlying window management system. These layers enable single-user applications to be shared transparently. The greatest drawback of the collaboration-aware approach is that all applications being shared must be specially designed or redesigned from scratch. In order to reuse the large amount of existing single-user applications, most users choose the collaborative-unaware

approach. However, to build collaborative-unaware systems to share single-user applications transparently is a great challenge [1].

In order to satisfy this demand, products and prototypes have been developed to support synchronous and asynchronous collaboration. A widely used example is the chat room. People in different places can chat with others synchronously through the Internet. There are also many frameworks to support group work on a shared workspace. For example, the whiteboard is a commonly used synchronous collaborative tool that lets users to work collaboratively through networks. Users can share files on the whiteboard and draw and annotate on it as well.

However, there are still a lot of problems with existing products, prototypes and frameworks. Some of these systems provide only simple tools, such as the chat room and the whiteboard. For instance, users cannot share Java programs transparently in the shared workspace in the TeamRooms [12] and CBE [19]. Most of them do not support latecomers. The session management in distributed networks is another challenge. Most existing products that support multiple sessions, such as Microsoft NetMeeting [20], heavily depend on central servers to handle session management. The quality of network connections and platform independency also needs to be considered.

These problems have limited the promotion of synchronous collaboration systems, so it is necessary to find solutions for them. Among various problems, we choose the most important one to be the objective of our project. They are the latecomer support, client synchronization and multi-session support. After adding these three tools, typical synchronous collaboration systems will become more complete. As more and more people begin to work in 3D world and VRML becomes more popular, the collaborative VRML viewer becomes another research point in our project.

1.2 Existing Problems

Since many collaboration systems were developed in the last decade, several shortcomings have been discovered during the use of these systems. In most existing systems, all the users have to start the session at the same time. Otherwise, they will see different views of the shared application. Therefore, latecomer support has been a problem for the promotion of tele-collaboration systems. Some collaboration systems support multiple sessions. However the session management depends heavily on centralized servers and it is not generic enough to be implemented in distributed networks, such as peer-to-peer networks. Moreover, almost no system considers the latency caused by network transmissions as a problem that may have impact on collaboration systems. In fact, latency may cause problems when clients' network transmission speeds are of great differences. For example, if one client connects to the server by an ATM connection and another client connects to the server through the phone line, events distributed by the server will arrive at the first client much earlier than at the second client. These two clients will definitely see different views of the shared application at the same moment.

As 3D applications have been used widely, demands for sharing 3D objects increase as well. VRML is a language used to define 3D objects. By installing a VRML plug-in or other additional software, commonly used browsers, such as MS Internet Explorer and Netscape Navigator, can be used to view VRML files. One of the important features of a transparent collaboration system is to share applications without any modification to files or special changes to the platform. Therefore, it will be more convenient to share VRML applications using ordinary browsers without installation of any plug-in or additional software.

1.3 Thesis Objective and Contribution

As demands for tele-collaboration increase, the requirements to collaboration systems have been increased too. Users are not satisfied with traditional synchronous collaboration systems, such as chat rooms and whiteboards. The objective of this project is to enhance and optimize typical transparent synchronous collaborative environments.

Some useful collaborative tools are added to these systems and make them as complete and generic as possible.

In this thesis, the development of several useful collaboration tools for transparent synchronous collaboration environments is described. First, the development of the latecomer support for synchronous systems is introduced. The implementation of generic latecomer support is a challenge to the development of typical synchronous collaboration systems. Second, the implementation of client synchronization for synchronous collaboration systems is presented. It is used to minimize the latency caused by network transmissions. When differences among network connections become big enough, latency may become a factor that influences the quality of tele-collaboration systems. Third, a special solution for multi-session support is introduced. Multiple sessions are supported by many similar systems, such as Mushroom [10] and LEVERAGE [11]. However, these solutions depend heavily on centralized servers. In the solution presented in this thesis, the lightweight multi-session support is proposed. It can be easily migrated to other network structures, such as the peer-to-peer networks. Finally, the implementation of a collaborative VRML viewer is described. The advantage of this viewer is that users do not need to install any plug-in or additional software to view VRML files.

1.4 Thesis Organization

This thesis describes the work of optimizing transparent synchronous collaborative environments by adding more tools to typical systems. The goal is to build such a system that makes virtual collaboration as real as possible.

In chapter 2, related knowledge about the development of collaboration systems and some similar work done by other teams are introduced. In chapter 3, system designs are presented in the order of latecomer support, client synchronization, lightweight multi-session support and collaborative VRML viewer. In chapter 4, the implementation of the above system designs is described in details. Chapter 5 covers the measurement and

evaluation of the whole system and each separated part. In chapter 6, a conclusion is given and some future work is mentioned.

1.5 Publications Resulting From This Research

Two papers have been submitted to conferences.

1. "Latecomer support and client synchronization for synchronous multimedia collaborative environments". This paper has been sent to The Fourth International Workshop on Collaborative Editing, ACM CSCW 2002, New Orleans, Louisiana USA.
2. "A multi-session synchronous multimedia collaborative environment". This paper has been sent to the Technical Symposium on Computer Science Education, ACM SIGCSE 2003, Reno, Nevada USA.

Chapter 2 Related Work and Background

2.1 Related Work

In this section, we will introduce some similar work in this field done by other teams. Some of them have latecomer support and some have multi-session support. We will introduce what these teams have done and the basic concepts and architectures behind their work. The differences between these works and our solutions, as well as the advantages of our solutions, are also presented following the introduction.

2.1.1 DISCIPLE

DISCIPLE (DIstributed System for Collaborative Information Processing and LEarning) is developed at Rutgers, the State University of New Jersey. It is a framework for sharing JavaBean applications in a real-time synchronous collaborative environment. Its generic collaboration bus provides a plug-and-play collaborative environment for both collaboration-aware and collaboration-unaware applications. The key component of DISCIPLE is the collaboration bus. It supports applications to get loaded and enables multi-user sharing. It requires no code modifications either in the underlying Java platform or in the applications. This is achieved by exploiting the Java delegation event

model, where the bus acts as a listener for application events. Figure 2.1 shows the architecture of the collaboration bus [2] [3] [4].

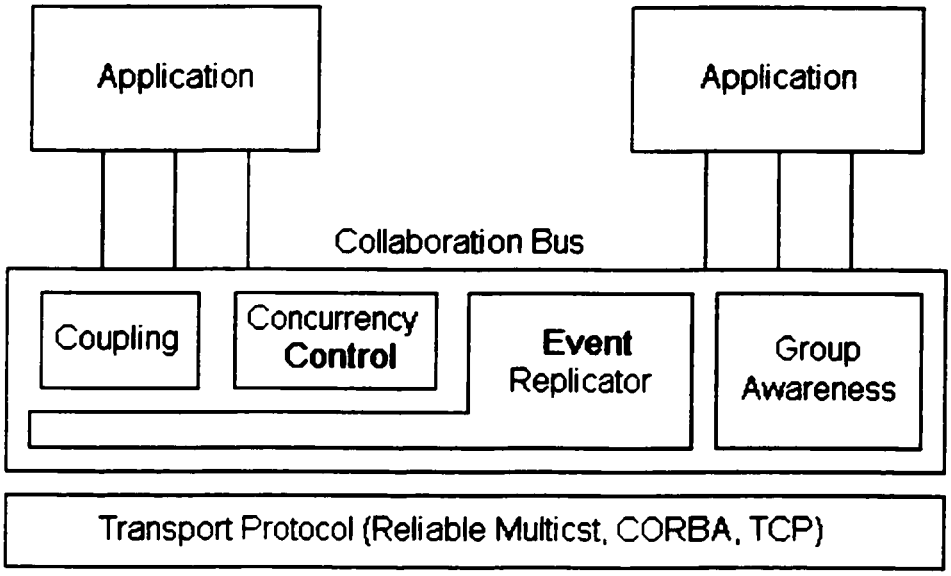

Figure 2.1 The architecture of DISCIPLE

DISCIPLE supports latecomers by implementing two alternative algorithms. The first one is the event logging and replay. In this algorithm, each site keeps a log of all the received events that change its application state. When a newcomer joins the session, he multicasts its presence and requests the current shared application state. All the active sites that receive this event reply with a confirmation message. The latecomer accepts only the first message and establishes a point-to-point connection with its sender and receives all the events that are logged in the sender's log. The algorithm is completely distributed and resilient to site failures. However, a disadvantage of this solution is that each site must keep an event log in memory [5].

The second algorithm is exporting the application state. In this algorithm, an interface is implemented for latecomer support. Before a site sends the current shared application state to a latecomer, it calls the interface method on the local application to retrieve its current state as an array of bits. The framework encapsulates the array in a predefined type of event and sends it to the latecomer. Serializing the application state can take a

significant amount of time for a complex application state. During this, other sites might generate events, so they still maintain the log. The algorithm's main advantage is that not every site needs to maintain an event log. However, this algorithm loses the previous algorithm's generality in the sense that the application must be aware of latecomer support and that latecomers will not know the session history [5].

DISCIPLE supports latecomers, but it does not support multiple sessions or client synchronization. In our solution, only one copy of the event log is kept in the collaboration server and this reduces the redundancy of the system. This is an improvement to the DISCIPLE system.

2.1.2 Habanero

Habanero is developed by the National Center for Supercomputing Applications. It is a software framework facilitating the construction of software for synchronous communication over the Internet. It also provides an environment for creating and participating in collaborative sessions and a set of tools, which utilize the framework and demonstrate many of its capabilities. Habanero is designed to allow collaboration among a small or large number of people with no limitation on the numbers or kind of collaborative software and hardware. It also enables construction of groupware systems and supports the transformation of single-user software tools into multi-user collaborative tools. Tools can be hosted on any hardware platform that supports Java, allowing session participation regardless of the participant's hardware [6].

To support latecomers, Habanero replicates applications across clients and shares all state changes among those clients. When a new client joins a session, it asks for information about running applications in the session. Then each application is sent enough information to completely replicate the important state being shared by the existing copies of that application. Habanero also ensures that all clients see the same state changing events in the same order, which results in applications appearing the same to all clients. This is implemented by marshalling and unmarshalling the hablet state. A hablet

is a Java applet program extended for multi-user collaboration. When a latecomer arrives, Habanero calls a marsallSelf method and the hablet marshalls itself to a marshall object. Then Habanero migrates this marshall object along with the hablet code to the latecomer's workstation. Then the migrated hablet code unmarshalls the previous hablet state from the marshall object. In addition, the migrated hablet needs to update the latecomer's user interface. Habanero also supports session record and replay feature, but this is a separate mechanism that is automatically supported by the system. Figure 2.2 shows the Habanero collaborative environment interface [7].

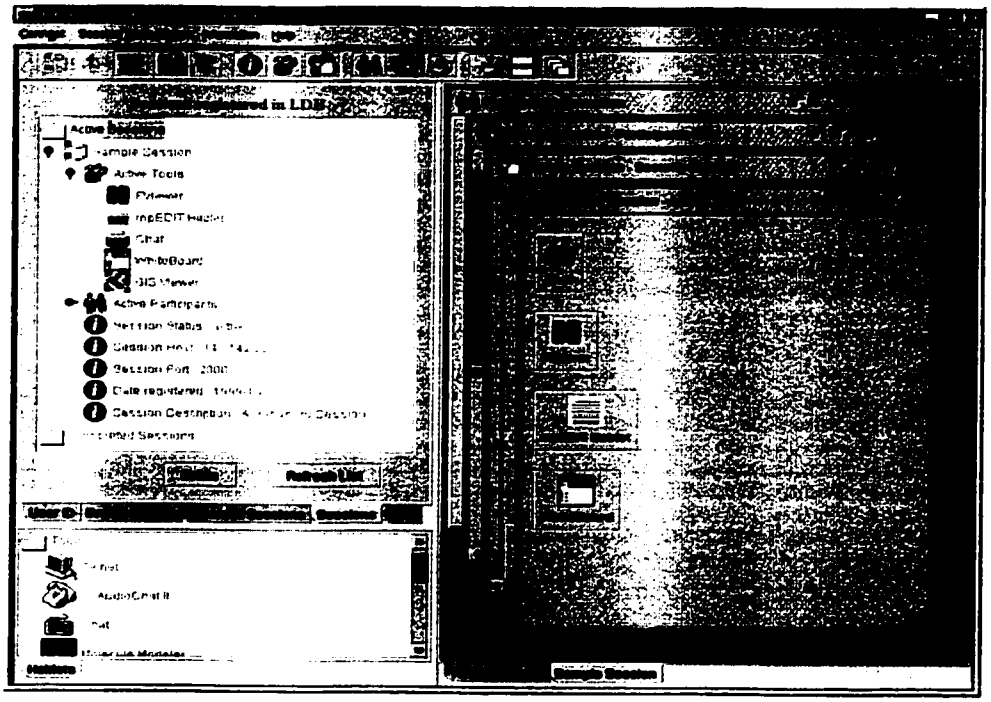

Figure 2.2 Habanero collaborative environment interface

Habanero supports latecomer and multiple sessions in a different way from our solution. However, Habanero replicates the whole shared application to latecomers which is not efficient. In our approach, only necessary states are transferred to latecomers. This reduces the workload of the communication server. However, the simplicity comes at the cost that a limit has to be set to the shared applications. The limit is that all the shared applications and applets should be serializable.

2.1.3 CHIME

CHIME is a Collaborative Virtual Environment developed at Columbia University. It has a modular architecture with a flexible GroupSpace Controller-based middleware. It uses a dynamic engine combined with themes to support on-the-fly world generation and supports real-time synchronization with one or more backend data sources. An event model is used to propagate changes and notifications effectively through the CHIME environment. This model follows a publish/subscribe model. When a change in the world occurs, an event is generated and transmitted to other users to serve as a notification of the change. Logged users can see the presence of other users and the changes they make to the system [8].

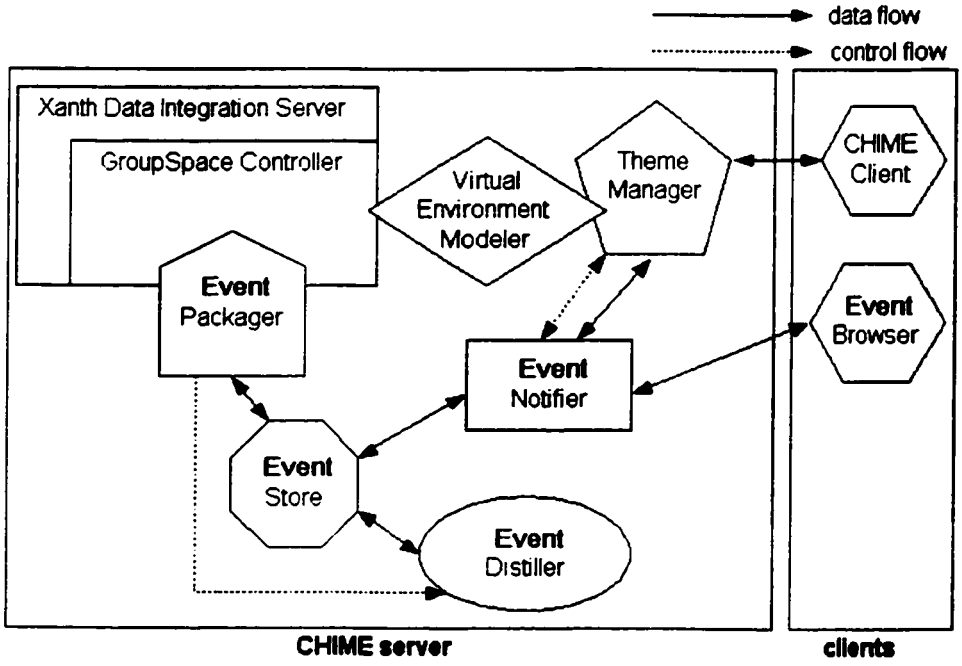

Figure 2.3 CHIME system overview

In order to solve the late-join problem with CHIME, three additional modules have been implemented: an event packager, an event distiller, and an event notifier. Figure 2.3 shows the system overview. The **event packager** collects all the events generated by CHIME. This is accomplished by having a GroupSpace service register itself for all events and listen appropriately. The packager then collects the events, compresses them,

and stores them. A persistent event database, called the event store, will contain both the preprocessed event data and distilled event data in an XML representation. The **event distiller** is running periodically and asynchronously. It processes and filters events logged by the event packager. The distiller processes events using heuristics coupled with a notion of the event hierarchy within CHIME without users' intervention [8].

The **event notifier** presents a user interface in the context of CHIME and CHIME's events. The interface has several representations. In a timeline-based representation, the timeline becomes a scrolling window while events are presented as bars. Clicking on a particular bar will bring up further details on the appropriate event. A zooming-based representation is an expansion on the timeline-based representation. Instead of scrolling, a user can click on a time zone and "zoom into" that particular zone to view further details at a more minute level. A user might click on a particular time range and zoom in again. Such a representation is useful in gleaning changes via a quick glance at the event timeline. This implementation is another kind of session recorder that records all the events happened to the system and lets users to view them in a user-friendly interface [8].

CHIME is a single session framework and has latecomer support for 3D worlds. However, its latecomer support is to replay recorded events. Our solution supports both the real-time state update and the session recording. These functions let latecomers reach the current state as soon as possible and know the events happened before they join the session as they wish. This is more complete than the CHIME system

2.1.4 Mushroom

Developed in the University of London, Mushroom supports the dynamic creation and management of Mrooms, which are shared environments for collaborative work and containers for shared objects. Mushroom supports centralized multiple session control, but it does not consider latecomer support. In Mrooms, users can share applications and information objects such as documents and whiteboards. Users also share tools and communicate with each other in the same Mroom. Mrooms contain representations of

collaborating users, the information objects on which they are working and links to other Mrooms. The Mushroom user interface offers a consistent view of each Mroom to all the users in the Mroom and provides operations for altering the contents of Mrooms. Objects may be dragged from one Mroom to another, with the default semantics that a copy is placed at the destination. Objects may exist in more than one Mroom. Figure 2.4 shows the architecture of Mushroom [9].

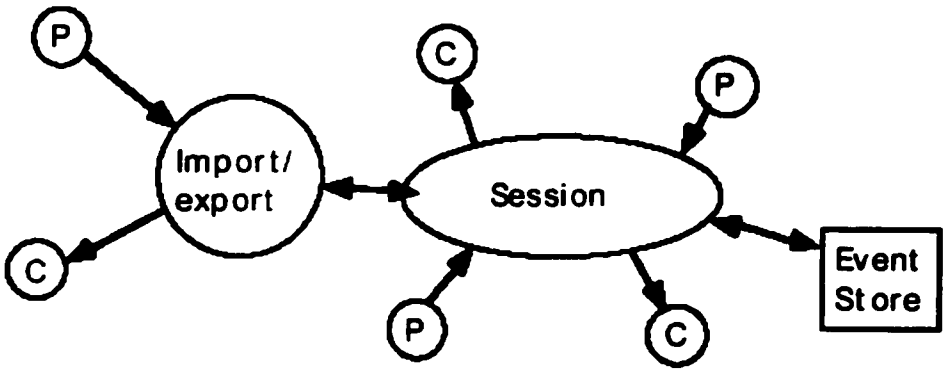

Figure 2.4 An event-oriented view of the architecture

(P = producer, C = consumer)

The term session is used to refer to a set of Mushroom processes that are maintaining replicas of some collections of objects in an Mroom. Each Mroom has a main session that manages the Mroom's root directory and representations of all the users in the Mroom. In general, other sessions are associated with an Mroom, in which a subset of the users is using a subset of the objects in the Mroom. Messages describing events are transmitted from the process initiating the event to all the members of sessions that are maintaining replicas of those objects. The event messages are delivered atomically to process in the persistence domain on a best-effort basis [10].

Mushroom supports centralized multiple session control, but it does not consider latecomer support. The multi-session support in Mushroom remembers not only environment variables but also hardware configuration and it is firmly integrated into the system. It cannot be moved to a distributed network.

2.1.5 LEVERAGE

LEVERAGE (LEarn from Video Extensive Real ATM Gigabit Experiment) is a collaborative research project partially funded by the European Commission's DG XIII ACTS program. The project aims to develop, implement and field trial a complete multimedia broadband network infrastructure to support collaborative, task-based foreign language learning between students remotely located. Collaboration is carried out by classical asynchronous tools, like e-mails or shared workspaces, and also by more advanced synchronous applications, like videoconferences, shared editors or blackboards [11].

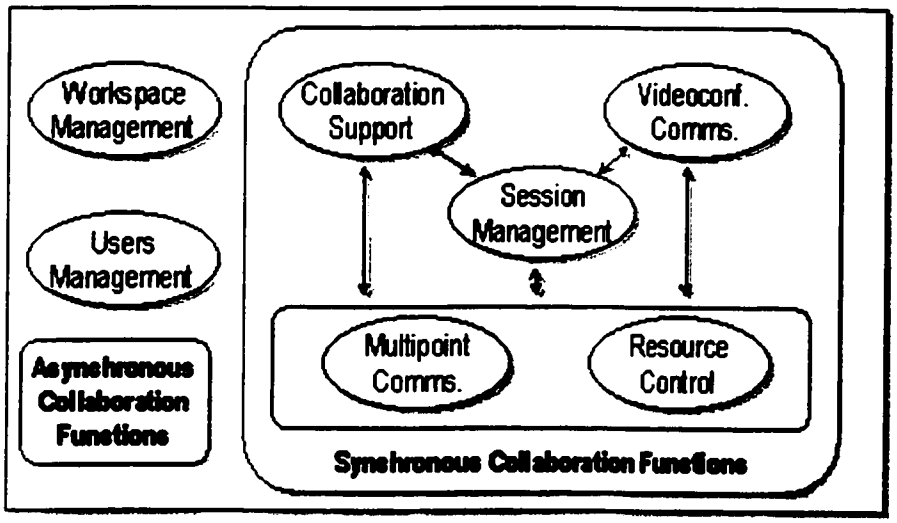

Figure 2.5 The functional architecture of LEVERAGE

Figure 2.5 presents the functional architecture of the LEVERAGE system and some relations between synchronous and asynchronous functions. The LEVERAGE Session Management Subsystem allows users to create, join, abandon or destroy collaborative sessions, as well as retrieving information about sessions. It is also in charge of organizing the communication and providing resources needed by synchronous applications [11].

From the user's point of view, the Session Management Subsystem is the entry point to the cooperative part of the LEVERAGE system. It offers a simple and user-friendly interface. From the technical point of view, it is a distributed application made of several modules that run on client machines and servers [11].

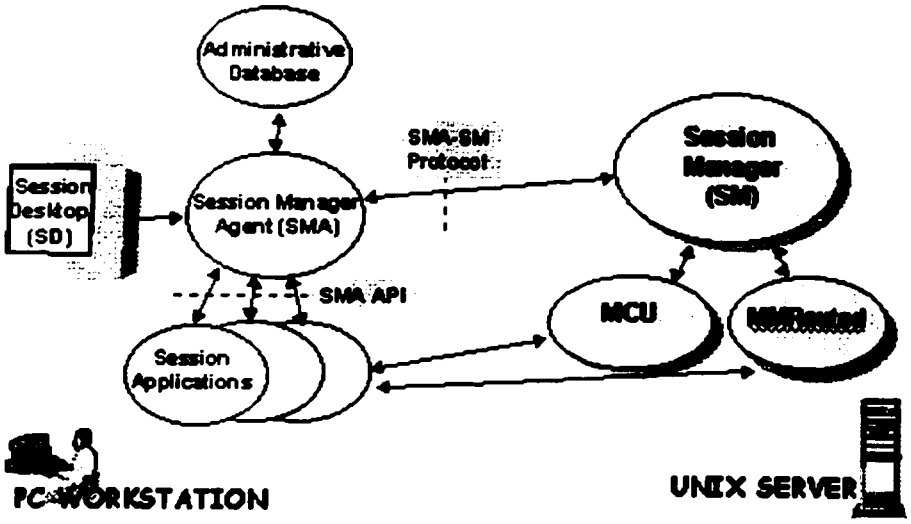

Figure 2.6 Session Management Subsystem Architecture

Figure 2.6 presents the simplified architecture of the LEVERAGE Session Management Subsystem, showing the different modules: Session Manager, which is the part of subsystem that runs on a UNIX server and centralizes all the information about sessions; Session Manager Agent, which manages the communications with the Session Manager and launches and coordinates cooperative applications on client machines; and Session Desktop, which is the user interface of subsystem as mentioned before. Modules running on UNIX server are implemented in C++. Modules running on clients and cooperative applications are written in Pascal using the Delphi development environment [11].

The LEVERAGE framework supports multiple sessions, but latecomer support is not included. The session management in the system is the key part to the server, so it is not suitable for other network architecture without centralized servers.

2.1.6 TeamRooms

TeamRooms is developed at the University of Calgary. It supports the team room concept for teams whose members work either co-located or at a distance. It combines both real-time and asynchronous groupware to provide the team with a "network place" serving as a locale for their collaborations. The system is highly customizable, allowing the team to

bring different tools into their electronic room on their needs. The system supports multiple sessions, but they use the word "room" instead of "session". The reach room contains both generic communication tools (a chat tool and a shared whiteboard) and any number of applets needed to support the group's work. When team members are in a room at the same time, they see not only each other but also each other's actions, both through immediate changes in the room's artifacts and through mechanisms such as multiple tele-pointers. As with real rooms, all artifacts in the electronic room persist even when no one is in the room. Figure 2.7 shows some applications in TeamRooms [12].

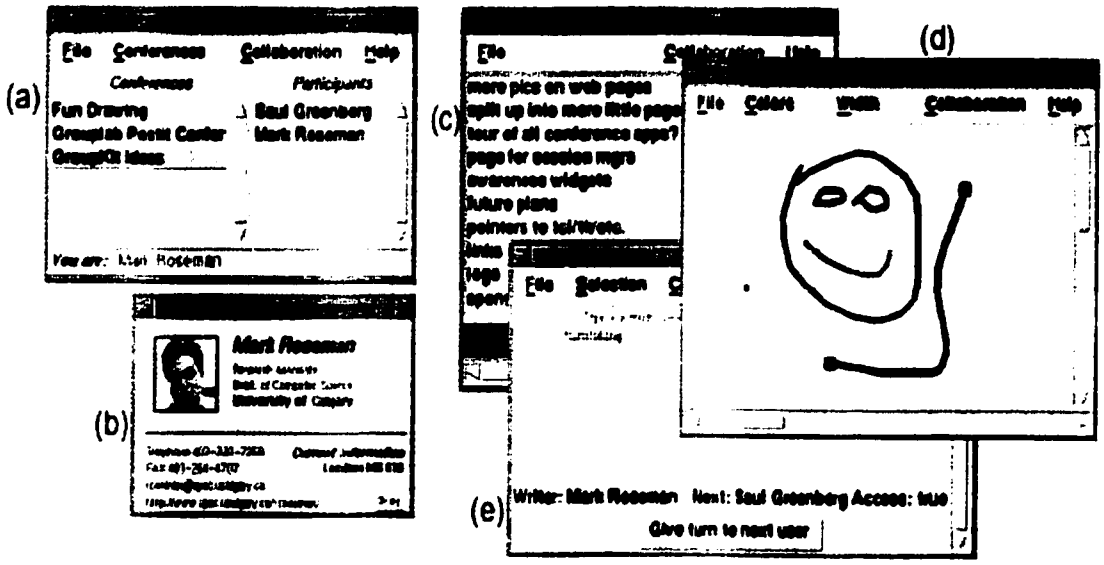

Figure 2.7 Applications in TeamRooms
including (a) the open registration session manager, (b) information on a user, (c) a brainstorming tool, (d) a shared whiteboard, and (e) s shared text editor.

TeamRooms consists of three separate components: the server, the user client, and the administration client. The server acts as a communication hub for clients to communicate with each other. A database of registered users is maintained. The server also maintains a persistence repository that states information about rooms on the server and their contents. When clients enter a room, information about the room is retrieved from the repository. Each server can hold any number of rooms. The administration client is whoever maintains the server. It can be used to create and delete accounts, change access permissions and provide tools to help manage the persistence repository [13].

Like Mushroom, TeamRooms also supports centralized multiple session control, but it does not consider latecomer support. They all use the room concept to manage system environment and resources. It is part of the central control system and cannot be removed from the system.

2.1.7 JCE

JCE (Java Collaborative Environment) is a framework for shared interactive multimedia applications in heterogeneous systems developed by the National Institute of Standards and Technology and the Old Dominion University. It intercepts, distributes and recreates user events that allow Java applications to be shared transparently. Their approach is based on a replicated tool architecture. Each site in the conference must have all the necessary objects for an application to ensure the correct operation with JCE. In addition to shared application, it also provides audio support as a standard feature [14] [15].

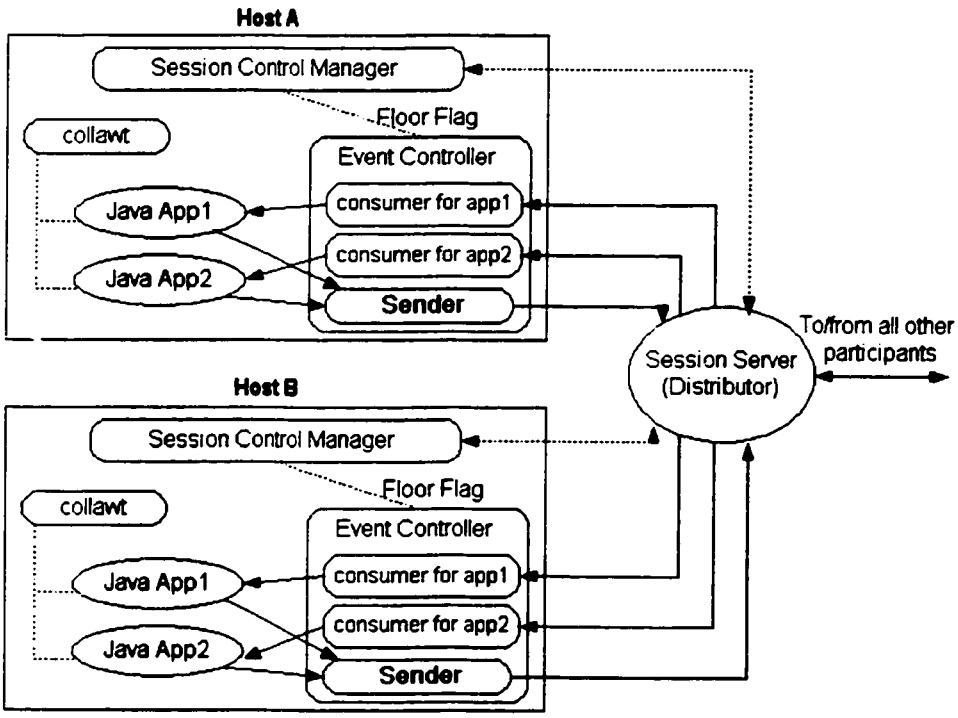

Figure 2.8 The overall system architecture of JCE

Figure 2.8 presents the overall system architecture and the relationship and communication paths among all processes of the system for a given conferencing session. JCE has three components: the Session Control Manager, the Event Controller and the Session Server. The **Session Control Manager** (SCM) provides a graphical interface to call, join or leave a session, to start applications, to request or release a floor, to invoke the audio system, or to start the replication editor that allows the user to create the profile for an application. The profile contains all the necessary information to be used to locate and replicate objects needed for a specific application before the application is started [16] [17] [18].

The **Event Controller** is composed of two processes: the event Sender and Consumer. Only one Sender exists for all applications. The Sender checks the intercepted event to determine whether or not it should be sent to the Session Server. The Consumer process for each application receives events redistributed by the Session Server from other participants, and posts them to the local instance of the application. The **Session Server** provides three distinct functions: distribution of all messages to all participants, group management for a given session, including joining or leaving a session, and server floor control management [16] [17] [18].

JCE has centralized multi-session support, but no latecomer support is implemented. Its session management is based on the client-server architecture and cannot be migrated to peer-to-peer networks.

2.1.8 CBE

The CBE (Collaboratory Builder's Environment) was developed at the University of Michigan Ann Arbor. It provides an infrastructure for building user-extendible Internet-based scientific collaborations. It allows users to add tools dynamically to shared workspaces and to move work dynamically between private and shared workspaces. In order to support dynamic reconfiguration of shared workspaces and to allow access over the Internet, CBE uses the metaphor of rooms as the high-level grouping mechanism for

applets. Rooms may contain users, applets and arbitrary data objects and can be used for both asynchronous and synchronous collaboration because their states persist across synchronous sessions [19].

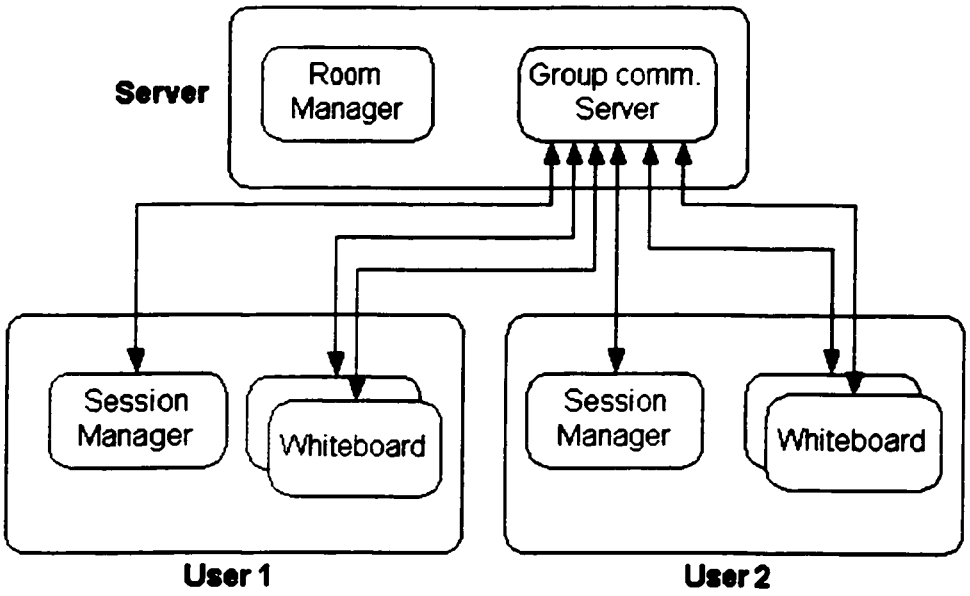

Figure 2.9 The run-time communication architecture in a collaboratory

Figure 2.9 shows the run-time communication architecture of a collaboratory. The Room Manager, the Session Manager and the Group communication server are the three main components of the system. Each user's workspace consists of one or more applets. They can communicate with each other through the group communication server. The **Room manager** and **Group Communication** are implemented as Java applications and run on a server machine. The **Session Manager** is implemented as a Java Applet running on clients' machines. CBE provides extendibility by allowing a collaboration to be constructed as a coordinated collection of group-aware applets. These applets are written by the collaboratory builder and they need to be group aware [19].

CBE supports centralized multiple sessions, but it does not have latecomer support. Just like JCE system, CBE cannot be used in the networks without centralized controllers, in this case, servers. Furthermore, all the shared applications need to be built by special software and they have to be group aware. These requirements have limited the use of this system.

2.1.9 Microsoft NetMeeting

NetMeeting is a powerful tool that allows real-time communication and collaboration over the Internet or corporate intranets. Users can communicate over a network with real-time voice and video technology. Users can work together virtually on any window-based program, exchange or mark up graphics on an electronic whiteboard, transfer files, or use the text-based chat program. NetMeeting supports multiple sessions based on a centralized server, but it does not support latecomers. In a session, only two users can have an audio and video conversion at a moment. Others have to use other tools to communicate.

NetMeeting functions as both a client and a platform. The NetMeeting client provides users the benefits of real-time audio, video and multipoint data conferencing. The NetMeeting platform also provides software API support so that software developers can integrate these conferencing features into their own products and services. To support this dual purpose, NetMeeting capabilities are built on the architecture of inter-networking components. Each component communicates with and passes data to and from the component layer above and below. This architecture, which is based on industry standards, ensures that manufacturers can easily develop products and services that build on the NetMeeting platform and interoperate with NetMeeting client conferencing features [20].

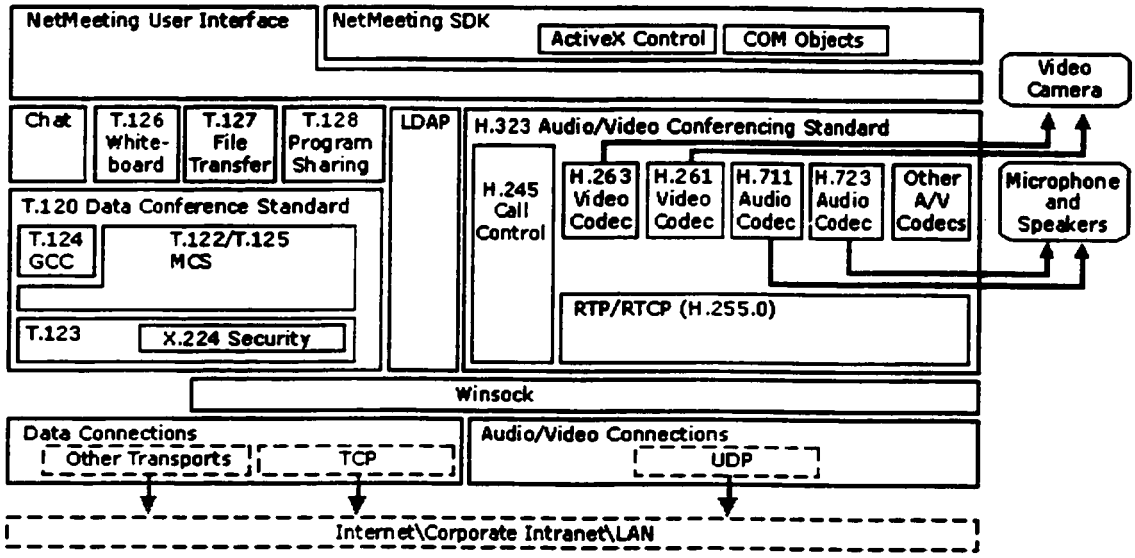

Figure 2.10 The architecture of Microsoft NetMeeting

At the core of the NetMeeting architecture is a series of data, audio and video conferencing and directory service standards. Figure 2.10 shows how these standards work together with transport applications, user interfaces, and the Window NetMeeting Software Development Kit (SDK) components to form the NetMeeting architecture [20].

NetMeeting is a very popular and powerful product. It supports multiple sessions based on heavy-duty centralized servers. The session management is simple and like the one we implemented in JASMINE. However, it is impossible to remove the centralized server from the system. NetMeeting provides latecomer support to videoconference users, but it does not support latecomers to other applications.

2.2 Literature Background

In this section, the overview background information of the thesis will be presented. The development of any collaboration system has to deal with handling operation sequences on shared objects, so the first concept to introduce is the concurrent programming for handling processes and shared objects. In order to provide a shared workspace for single-user applications, it is necessary to create an intermediate layer to support transparent

collaboration. A shared window architecture is an efficient way to provide such a layer between applications and the underlying window management system. Therefore, the concept of shared window architecture will be presented secondly. The next part is some related information about the development environment on which we work, including JASMINE and JETS. Latecomer support, client synchronization and multi-session support are implemented on JASMINE and the collaborative VRML viewer is implemented on JETS. Blaxxun3D is the VRML parser used for the collaborative VRML viewer. Finally, some brief information about VRML will be given.

2.2.1 Concurrent Programming

A concurrent program consists of processes and shared objects. A process is a sequential program executing on some processor and shared objects are built using shared memory or a computer-communication network. In order to cooperate, processes must communicate with each other. During the communication, one process may influence another by writing and reading shared variables or by sending and receiving messages. Both ways may involve delays. Time may elapse between the time a shared variable is written and the time it is read, and time can elapse between the time a message is sent and the time it is received. This delay means that information obtained by a process might reject a past state of the sender [21].

Using a shared object to communicate is possible only if the shared object is read after it is written. Reading the object before it is written may return a meaningful but erroneous value. Therefore, communication between asynchronous processes must have some kind of synchronization. Two ways of synchronization are widely used in concurrent programming. The first, mutual exclusion, groups actions into critical sections and blocks execution for a critical section in one process whenever a critical section from another process has started but not completed. The second one, condition synchronization, blocks a process until the system state satisfies some specified conditions. This is useful to order events that occur at different processes [21].

A program state associates a value with each variable. Variables include those explicit variables declared by programmers and implicit variables that cannot be read or written in assignment statements. The program counter for a process is an example of an implicit variable. It presents the next action to be executed by that process. Its value changes after the execution of this action. A shared communication channel is another example of an implicit variable. Sending or receiving a message is usually the only way to change the value of this implicit variable [22].

2.2.2 Shared Window Architecture

The shared workspace has been considered as a significant feature of synchronous concurrent environments. An efficient way to provide shared workspaces is the Shared Window System (SWS) lying between shared applications and the underlying Window Management System (WMS). With SWS, single-user applications can be shared transparently among multiple participants without any modification [1].

The basic idea of the Shared Window System is to catch events in WMSs and applications and distribute events among all participants. In some WMSs, the client-server model and a standard network protocol are used to transmit messages. These are called network-aware window systems. On the other hand, network-unaware WMSs ignore potential tele-cooperation and result in high-degree interlocking among applications, operation systems and I/O devices. The network awareness will significantly affect the implementation complexity of the requests/events intercepting [1].

A Shared Window System is a mediate layer between applications and WMSs to provide application-sharing services. Therefore, the capability to redirect request/event streams is essential. There are three ways to build SWS according to the differences of intercepting stream data. The first one is the event-sharing paradigm that focuses on synchronizing of the events sent to or from shared applications. The second one is the UI-image-sharing paradigm that detects and refreshes the image content changing of the user interface of

shared applications. The last one is the request-sharing paradigm that intercepts, distributes and reproduces all output related requests and input events [1].

1. Event-sharing paradigm

The basic requirement of the event-sharing SWS is that all participants must run the same application on their machines. By collecting and synchronizing input events, applications will behave in the same way. The implementation topics of event-sharing SWS include intercepting, collecting and rerouting of events. Therefore, an event interceptor module is used to catch events coming from local or remote WMS. An event reproducer module is used to provide filtered events to applications. A shared activity manager module is used to collect all intercepted events and distribute them following certain floor control rules [1].

Because shared applications must run on all participants' machines, the pure event-sharing SWS is naturally a replicated system. Filtered events are manipulated instantly by local applications, so event-sharing SWSs have very short responding time. Furthermore, typical event packets are very small, so they consume very low network bandwidth. However, pure event-sharing SWSs suffer from certain drawbacks. For example, applications been shared must be deterministic. An application is deterministic if outputs of the application are determined by certain user inputs. Typical non-deterministic applications are those using random numbers or local system status to display certain information on UI. If non-deterministic applications are shared, event-sharing SWSs cannot guarantee that same status can be reached by only distributing same input events. Moreover, some advanced collaboration features, such as latecomer support, spontaneous application sharing, and cross-platform sharing, are difficult, even impossible to implement [1].

2. UI-image-sharing paradigm

Another way to implement shared workspaces is to distribute the visual content of applications' user interfaces by hard copying all UI images. It is called the UI-image-sharing paradigm. In this case, shared applications are only executed on the provider site. Other participants (called sharer) receive only the UI updated images from the provider and they do not need to run shared applications. Therefore, the UI-image-sharing SWS is naturally a centralized system. The implementation criteria of UI-image-sharing SWS include finding out the proper time to capture UI images, reconstructing updated UI images, and minimizing the transmission bandwidth [1].

Because the UI updating of an application is achieved by sending output requests to underlying WMS, intercepting certain types of output requests can be used to trigger the UI grabbing function. The intercepted requests should be by-passed to WMS without any modification. On the provider site, a UI-image monitor model is invoked to gather the shared applications' UI updating. An event reproducer model is invoked to provide filtered input events to the shared application. On the sharer site, a UI-image builder model is invoked to reconstruct the original UI of the shared application. The event interceptor model is needed in both sites to collect all input events following certain floor control rules [1].

Since input events and UI image blocks are transmitted, it is possible for UI-image sharing SWSs to support advanced collaboration features, such as latecomer support, spontaneous application sharing and cross-platform sharing. Furthermore, UI-image sharing SWSs can eliminate "deterministic shared application" problems that greatly harm the pure event-sharing SWS. However, UI-image sharing SWSs need far more bandwidth than pure event-sharing SWSs do [1].

3. Request-sharing paradigm

Another way to achieve a shared workspace is to multiplex applications' output requests directly. It is called the request-sharing paradigm. Request-sharing SWSs are also centralized systems. Shared applications only run at the provider site and SWSs residing

on sharers generate "pseudo" application objects by reproducing those intercepted output requests. On the provider site, the output requests are caught and analyzed by the request interceptor model, and passed to the local request reproducer model to complete original tasks. All request related to local information must be further extracted and packed with requests themselves into shared packets and distributed to all sharers. Sharers receive shared packets, unpack them into original form, and send them to the request reproducer model for local reconstructing [1].

The implementation of output requests depends highly on underlying WMSs. For example, system object identifiers are only meaningful to local systems and must be re-mapped when reconstructed on other sites. In some WMSs, especially those network-unaware systems like MS Windows, requests are tightly coupled with local operating systems. It could be a great challenge to implement a request-sharing SWS on such WMSs. The network bandwidth consuming can be more effective than that of UI-image sharing paradigm [1].

2.2.3 Introduction to JASMINE

JASMINE (Java Application Sharing in Multi-user INteractive Environment), developed jointly by the University of Ottawa and the Darmstadt University of Technology, enables users to share Java applications or applets in real time. These applications or applets can be pre-loaded or brought to the session live. The system also provides basic moderation control and enables diverse views of the same visualization in a moderation session, in which the moderator can see more than others [23].

JASMINE is based on a Java Events Delegation Model, which provides a standard mechanism for a source component to generate an event and sent it to a set of listeners. It also allows sending events to an adapter, which works as an event listener for the source and as the source for listeners. Since event handling is critical in a event-sharing collaborative system, this mechanism makes the implementation much more flexible and makes event handling much easier. The idea behind JASMINE is that all the events

happening to the GUI of an application or applet on the client side can be caught, distributed and reconstructed. As the result, Java applications or applets can be shared transparently. This type of collaboration allows users to interact in real time [23].

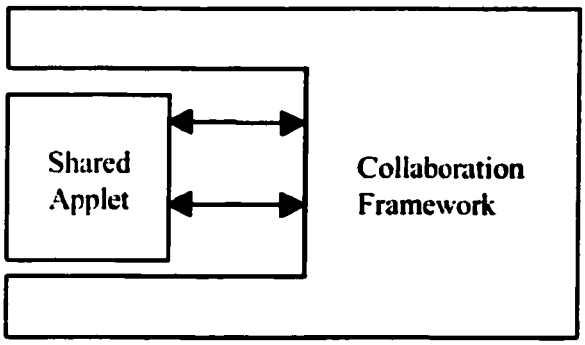

Figure 2.11 The concept of JASMINE

Figure 2.11 shows the concept of JASMINE. It wraps around the application to be shared. The framework listens to all the events occurring on the GUI of the application and distributed these events to all other participants and then reconstructs events there. According to the information we introduced in the chapter 2, JASMINE uses the event-sharing paradigm to implement the shared window system. Therefore, it comes with advantages and disadvantages of this approach. It is easy to implemented, having short corresponding time and consuming low network bandwidth. However, it will not function when it shares non-deterministic applications [24].

2.2.3.1 Introduction to JASMINE Server

JASMINE uses a client-server approach. The server is responsible for receiving events and distributing them to other users in the session. The communication module is implemented on top of TCP because the reliability is critical for a collaborative system. This approach makes the system structure clearer and simpler, but the scalability becomes a major problem. As the number of connected users increases, the workload of the server increases in a non-linear way. This will definitely cause a bottleneck on the server and reduce the performance of the whole system. Therefore, JASMINE is suitable for small to medium size group collaborative sessions [24].

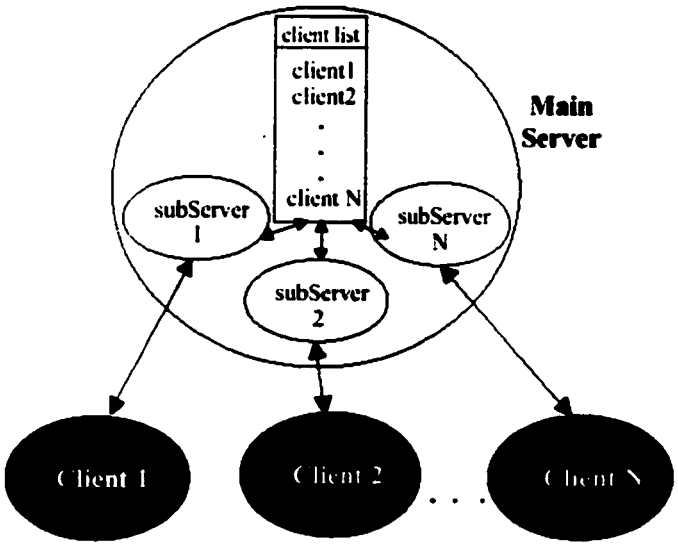

Figure 2.12 JASMINE server-client architecture

JASMINE uses a multithreaded server as shown in the figure 2.12. When a client connects to the server, the main server starts a sub-server for this client and it handles only incoming messages from its own client. When the sub-server receives the update messages from its client, it sends these messages to all other clients in the session. Only one client can interact with the shared application at a time and other threads are simply waiting for their turns. This will not consume too many resources. Besides handling incoming events and messages, the server also provides other services, such as the session moderation and floor control [24].

There are several important classes in the JASMINE server. The class **ServerLogin** is the start point of the server software. Users can set the port, name, moderation at this point. The class **JasmineServer** is the main part of the JASMINE server. It is responsible for creating a socket and listening to it for any message from clients. Whenever a client connects to the server, it starts a **subServer** for this client and the subServer will handle all the income messages from this clients and send messages to other clients who have also connected to the server. The **Mutex** class is for the floor control that sets permission to only one client to operate at one moment.

2.2.3.2 Introduction to JASMINE Client

The JASMINE client is responsible for capturing events, sending events to the server, receiving events from the server and reconstructing events locally. It is a Java application that consists of four important components:

- Collaboration Manager
- Listener Adapter
- Component Adapter
- Event Adapter

The **Collaboration Manager** is responsible for communication between clients and the server and it provides GUI as well. The **Listener Adapter** implements several AWT listeners. After catching an event, the Listener Adapter converts the event to a remote one and forwards it to the Collaboration Manager. The **Component Adapter** maintains a list of references to the GUI components of all applications and applets. This list is created in the same order on each client, so each component has the same reference number on all clients. The **Event Adapter** works opposite to the Listener Adapter. It converts remote events to local events and applies them to corresponding components [23].

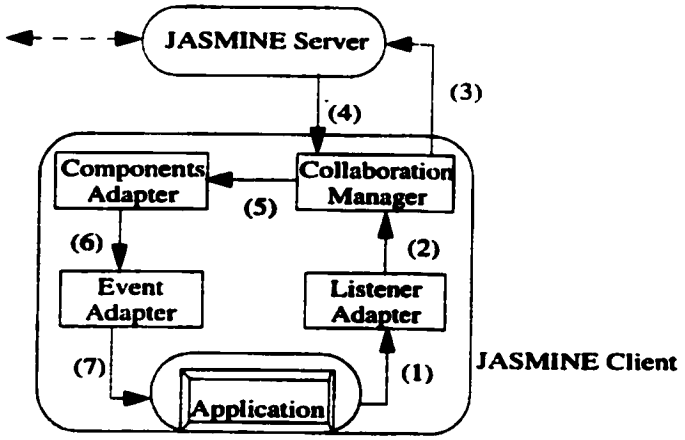

Figure 2.13 JASMINE data flow diagram

The figure 2.13 presents the data flow diagram of the client side architecture. There are two main data paths in the system. The first path is labeled with 1, 2, and 3. Any Java event occurring in a Java application is caught by the Listener Adapter. If the event is a

local event, the Listener Adapter converts it to a remote event and forwards it to the Collaboration Manager. Then the Collaboration Manager sends the remote event to the JASMINE server. The second path is label with 4, 5, 6, and 7. The remote event received by the Collaboration Manager is forwarded to the Component Adapter. The Component Adapter gets the information about the source of the event and sends it with the event to the Event Adapter. The Event Adapter converts the remote event to a normal AWT event and dispatches the event to the corresponding component [23].

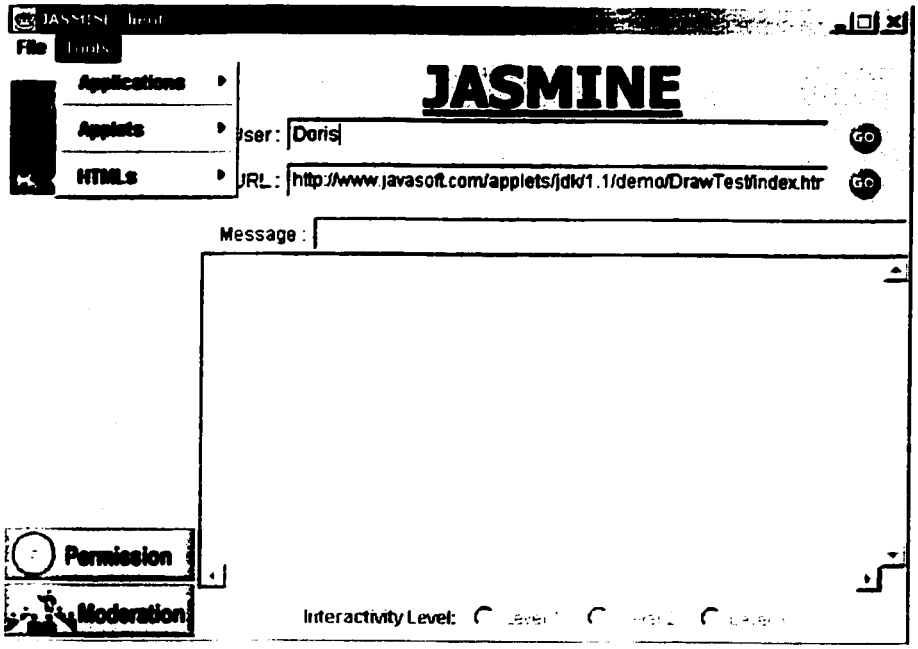

Figure 2.14 The interface of JASMINE client

Some classes are implemented in the JASMINE client. The class **ClientLogin** is the start point of the client software. Users need to fill in the IP address of the server to connect to the server. The class **CollaborationManager** is the main part of the client software. It provides a GUI and it is responsible for communications between clients and the server. The class **ListenerAdapter** converts the event to a remote event and forwards it to the CollaborationManager. The class **ComponentAdapter** maintains a list of references to the GUI components of all applications and applets. The class **EventAdapter** converts the remote events to local events and applies these local events to corresponding components. The class **JasmineJMFPlayer** is used to play JMF supported video files. The class **DynamicMenuItem** enables users to load client interface menu items

dynamically from the client property file. The class **CollaborationBrowser** is a simple browser to display web page and it can be used to load Java applets. Classes **PermissionDialog**, **PasswordDialog** and **InfoDialog** are prompted to let users enter required information. All the events classes are used to transfer standard AWT events to remote computers.

2.2.4 Introduction to JETS

JETS (Java Enabled Telecollaboration System), developed at the University of Ottawa, is a client-server system that permits sharing of Java applets. Since JETS uses core Java packages, users do not need to install any additional Java classes on their systems. This allows any user to access JETS and share applets with a java-enabled browser. Because of security restrictions, applets are allowed to establish network connections only to the machine from which they are downloaded. This issue forces JETS to use a central server, so applet clients can exchange data and multimedia documents through the server. For the client-server communication, JETS uses TCP/IP and UDP/IP sockets. Obviously, scalability becomes an issue since too many clients would overwhelm the server, so JETS has been designed in such a way that scalability becomes a hardware issue. Furthermore, JETS makes it possible to use more than one computer to perform duties of the server. Figure 2.15 below shows how JETS works [25].

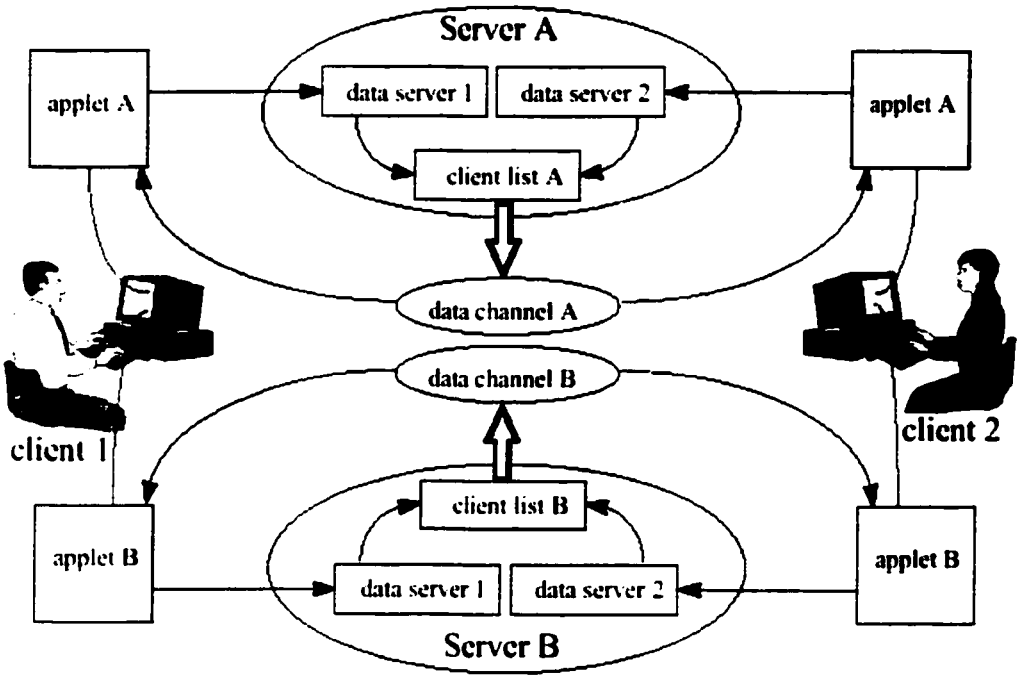

Figure 2.15 The architecture of JETS

JETS is a groupware toolkit, written fully in Java, that offers an API which a developer can use to create multimedia applets for collaboration. The API consists of mostly server side classes that provide the necessary functionality and services for collaboration. It uses a series of semaphores on the server to provide locking [25].

The whiteboard is an interactive space where clients in a virtual session can share pictures, Microsoft PowerPoint slides, text, video and drawings. Users can annotate on images and start a discussion. The built-in locking mechanism of JETS is used to avoid modification of the same objects at the same time by more than one user. The simplest way of interacting on the whiteboard is through text. Clients can use the chat area to communicate. All other members who have input access to the whiteboard will see the originator of the message followed by the message itself. Another way of interacting through the whiteboard is by drawing. To do this, the user simply chooses a color from the color template and draws by keeping the left mouse button down and dragging the mouse around the screen. A client may paste a picture found in the archive. Any member who has full access can freely comment or draw on the picture. A client may also start a

slide show found in the archive. Any member who has full access can freely comment or draw on the slide show as well as go to the next or previous slide [25].

A very useful feature of JETS is its ability to play ITU-T H.263 compliant video on the whiteboard. That is accomplished by using jStreaming's API. When a user opens a video file and starts playing it, video data are streamed down to all participants, decoded in real-time, and displayed on their whiteboards [26].

Another sample applet is a simple shared 3D viewer that permits real-time collaborative interaction with simple VRML objects. The applet brings from the server simple VRML 1.0 files and displays them in the wire frame mode. A user can then collaboratively interact with 3D objects, with all the rotation, moving, and zooming reflected on all participants' screens [26].

JETS also implements a management system that enables monitoring of the session. One of the session clients has to log in as a moderator. Once the session management is established, initially only the moderator has the right to access the shared whiteboard. Other clients have no access. Every session client can ask the moderator for access permission. Once the moderator approves a session client's request, this client becomes a session participant, and gains the right to access the shared whiteboard. Any session participant can put his/her notation on the shared whiteboard. The moderator has the right to revoke access privileges to any client at any time [26].

2.2.5 Introduction to blaxxun3D

Blaxxun3D is a product of the blaxxun Inc. in Germany. Blaxxun3D can be used for e-commerce, education, data visualization, entertainment, web animations and other applications. Like any other Java applet, it can be integrated into a standard web page. It is completely transparent to all web page visitors and is displayed with other page content. It requires no download or installation and loads automatically when a visitor opens a web page that contains the applet. All authoring tools that generate VRML code

can be used for producing blaxxun3D content. Users can extend blaxxun3D's functionality on their own using its built in API, connect blaxxun3D to their database or use Java's network capabilities.

Blaxxun3D achieves good compromise between minimum files size and maximum functionality. Although it's only 55KB small, blaxxun3D implements a major part of the VRML97 specification. However, it is possible to add users' desirable additional functionality that has not been implemented in the core blaxxun3D applet. To realize such extra functionality, blaxxun3D offers a powerful API. It allows users to access the render applet and extend its functionality on their own. This API is very similar to the External Authoring Interface (EAI) provided by all major plug-in-based VRML browsers. Generally, all blaxxun3D extensions are done the same way: user creates a second applet that connects to blaxxun3D and implements the additional functionality. The second applet is embedded in the same HTML file as blaxxun3D[27].

Once the connection to the render applet has been successfully established, users can connect to any node in the VRML file. Whenever users want to manipulate any node displayed in blaxxun3D, users have to connect to them. About the interaction, blaxxun3D offers two event observers, an AWT event observer that notifies users about AWT events and a browser observer that informs users about render events [27].

2.2.6 Introduction to VRML

VRML (Virtual Reality Modeling Language) is a language for describing three-dimensional image sequences and possible user interaction. Using VRML, we can build a sequence of visual images into web settings with which a user can interact by viewing, moving, rotating, and other interacting with an apparent 3D scene [38].

VRML was designed to allow a 3D world to be delivered over the World Wide Web. VRML files are standard text files that can be interpreted by browsers. Like HTML, VRML has been designed to be platform independent and to work over low bandwidth

connections. While HTML files describe a 2D page containing various text constructs, VRML files describe a 3D space, or "world". Using a VRML browser, users can explore a 3D world, zooming in and out, moving around and interacting with the virtual environment. This allows complex 3D graphics to be transmitted across networks without very high bandwidth [28].

VRML files can be parsed by a browser. The browser may be equipped with a plug-in for a web browser or a helper application. The scene graph is composed of nodes. These nodes may have routes between them, above and beyond the scene graph hierarchy, which define the possible interactions of one node with another. Nodes have fields that define the actual value associated with them. Within the scene graph, certain special nodes allow fields to be pushed and popped to control certain world parameters [28].

VRML is intended to make virtual worlds available over a shared computer network, such as the Internet. Although VRML is really just a computer language that describes three-dimensional objects, it has come to signify virtual reality's arrival on the Internet. The line between virtual reality and VRML is becoming gradually fuzzier. Recent additions to the VRML specification will allow users to interact with objects in the world. Interaction has always been the key ingredient of a virtual world [28].

Chapter 3 System Design

3.1 Latecomer Support

3.1.1 Background Introduction

Synchronous collaboration environments let distributed users work in a shared workspace. However, most existing products or frameworks do not support latecomers. For instance, Microsoft NetMeeting, a widely used synchronous collaborative system, does not support latecomer joining. All the clients have to start the session at the same time. Otherwise, they may see different views of the ongoing session. In the real world, the number of users in a collaboration session changes dynamically. Some users start a collaboration session. Later on some users may join the session while some others may leave it. Latecomer support allows users to join a collaboration session already in progress. The essential part of latecomer support is to create a new user interface that shows the current shared application state for latecomers [29].

3.1.1.1 Introduction to Generic Latecomer Support

Latecomer support is a challenge for distributed interactive systems since each client maintains a local copy of the application's share state. Without information about the current state of the running application, it is impossible for a latecomer to join an ongoing session. In order to let latecomer join an existing session, it is necessary to know some information related to the shared application. Since a large part of the state of the shared application may not be relevant to the latecomer immediately, first we need to know which part of the current state latecomers need. Second, the latecomer should receive the right information at the right time in a right way [30].

3.1.1.2 Existing Latecomer Support Algorithms

Existing algorithms can be divided into two categories. In the first one, latecomer handling depends on transportation protocols, such as the Scalable Reliable Multicast protocol (SRM) [31]. All the data packets from the beginning of the session are stored. The late coming application can reconstruct the current state according to the start state and these stored packets. However, protocol level algorithms have some disadvantages. First, it is not efficient to transfer all the transport packets because most of the transmission information may be irrelevant to the application. Second, some states cannot be reconstructed by using the transport packets. Algorithms in the second category are realized at the application level. In order to avoid problems related to the protocol level latecomer algorithms, most existing solutions are focusing on the application level approach [32].

There are several approaches to realize latecomer support at the application level:

- Centralized approach: It needs a server to handle all the latecomer requests. When latecomers log in, they send requests to the server. Then the server asks someone who is already in the session to send the current state to the server. After receiving the current state, the server distributes it to latecomers [32].
- Distributed approach: There is no need for a central control server to handle requests. When latecomers want to join a session, they ask whoever is in the

session to send the current state to them since the clients who are already in the session are supposed to have the same current state. In this case, messages do not need to be sent through the server. Connections are established between different clients [32].

- Hybrid approach: This is combining the above two approaches. A server is needed to handle all the latecomer requests and choose the source of the current state, but the update of state occurs only between clients.

The essential part of latecomer support is to create a new user interface for the latecomer and to show some parts of the shared application state [29]. How this is accomplished can vary. The exact part of the state shown may depend on the latecomer's role in the collaboration. Moreover, before displaying the state, latecomer support may also show a quick animation of how the user interface reaches its current state.

In our work, we choose the centralized approach. One reason is that the platform on which we are working is built upon a client-server architecture. However, it is very easy to migrate the solution to a hybrid approach since the state transmission is already through a new created socket and I/O streams. Another reason for choosing the centralized approach is to reduce the storage space for the event log. Using the centralized approach, clients do not need to maintain copies of the event log. All the events are stored in the server and can be sent to any client at any time. The system we use to implement latecomer support is JASMINE (see section 2.2.3).

3.1.2 Design of Latecomer Support for Transparent Shared Environments

3.1.2.1 Algorithm

The most important part of latecomer support is the ability to recognize and transfer the current state. As we mentioned in the concurrent programming section (see section 2.2.1), the program state associates values with variables. What we need for latecomer support is

to identify what variables are needed and what their values are. Then we can transfer these variables together with their values to latecomers and update latecomers' states.

The algorithm we use is based on a proposed latecomer support from a team at the University of Ulm in Germany. They have developed a simple latecomer support for shared Java applications and applets in JASMINE [33]. According to their solution, the current state is serialized into object streams and sent to latecomers through the server. Latecomers then reconstruct the user interface. However, their solution has some weaknesses. First, it supports only Java applications and applet, but not JMF players. Second, their algorithm does not consider the floor control and moderation. Third, their system has some problems with the implementation of the state serialization and transmission. Forth, their system is not stable because they did not handle the component reference list correctly. We optimize the algorithm and implementation and add more functionality to the system.

As a synchronous multimedia collaborative system, JASMINE supports sharing not only Java applications and applets transparently, but also JMF players. Since the system shares Java applications and the JMF player using different mechanisms, it is necessary to design and implement latecomer support for them in different ways.

3.1.2.2 Design of Latecomer Support for Java Applications and Applets

In order to implement the latecomer support for Java applications and applets, we need to set some constraints to the system:

- The solution is based on applications, including Java applications, Java applets and the JMF player.
- All the clients in the session, including latecomers, should have classes they want to share in their local machine.
- All the shared classes must be serializable.

Figure 3.4 shows the algorithm for the latecomer support for Java applets and applications in a transparent shared environment.

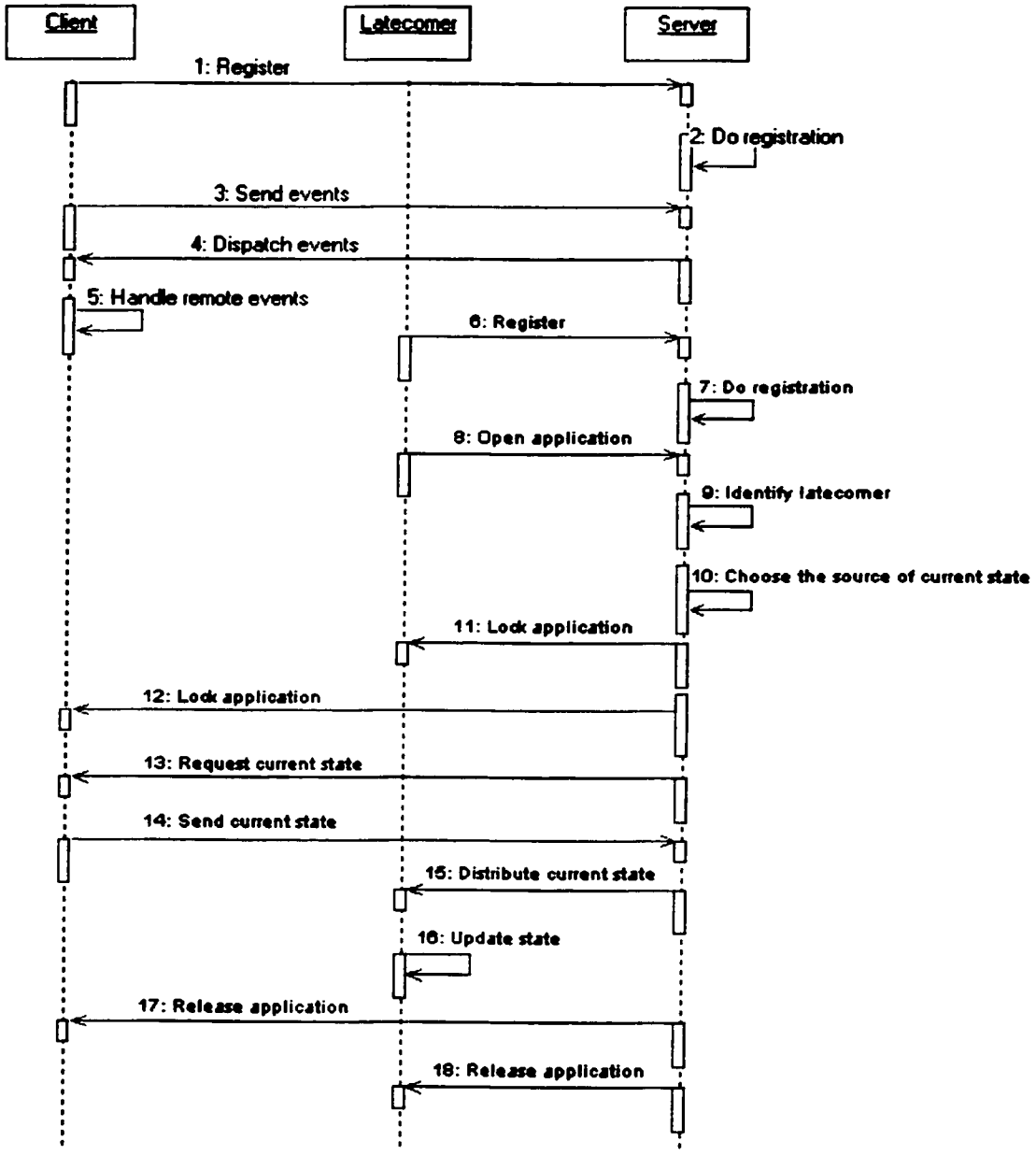

Figure 3.1 The algorithm for the latecomer support for Java applets and applications

Here is the main scenario for this case: When a client starts the client software and logs into the system, the server does not identify if the client is a latecomer or not. As described in the figure 3.1, only when the latecomer starts an application, like opening a

drawing board, the server checks if the client is a latecomer for this application. If the client is a latecomer, the server chooses the source of the current state and sends a lock message to both the latecomer and the source of the current state. On the latecomer side, after getting the lock message, the client locks the application, sets a connection to the server, sends a ready message to the server and waits for the transfer of the current state. On the source side, the client locks the application, creates a connection to the server, sends a ready message to the server and waits for a response from the server. Once the server gets a ready message from both sides and finishes connection setting for both sides, it forwards a ready message to the source and waits for the transmission. After the source receives the ready signal from the server, it starts to transfer the current state to the server and the server passes data to the latecomer without any modification. After the latecomer receives the current state, it starts to update its state and release an unlock message to the server. Once the server receives the release message from the latecomer, it forwards a release message to the source. When the source catches the release message, it unlocks the application and continues the session.

3.1.2.3 Design of Latecomer Support for the Media Player

In order to support latecomers for the media player, it is important to know the file that is playing and the frame number of the moment when the latecomer joins the session. Since each client maintains a copy of media files that are going to play, the latecomer support for the JMF player focuses on transferring the media file name and frame number to the latecomer. Figure 3.2 shows the algorithm for the latecomer support for the JMF player.

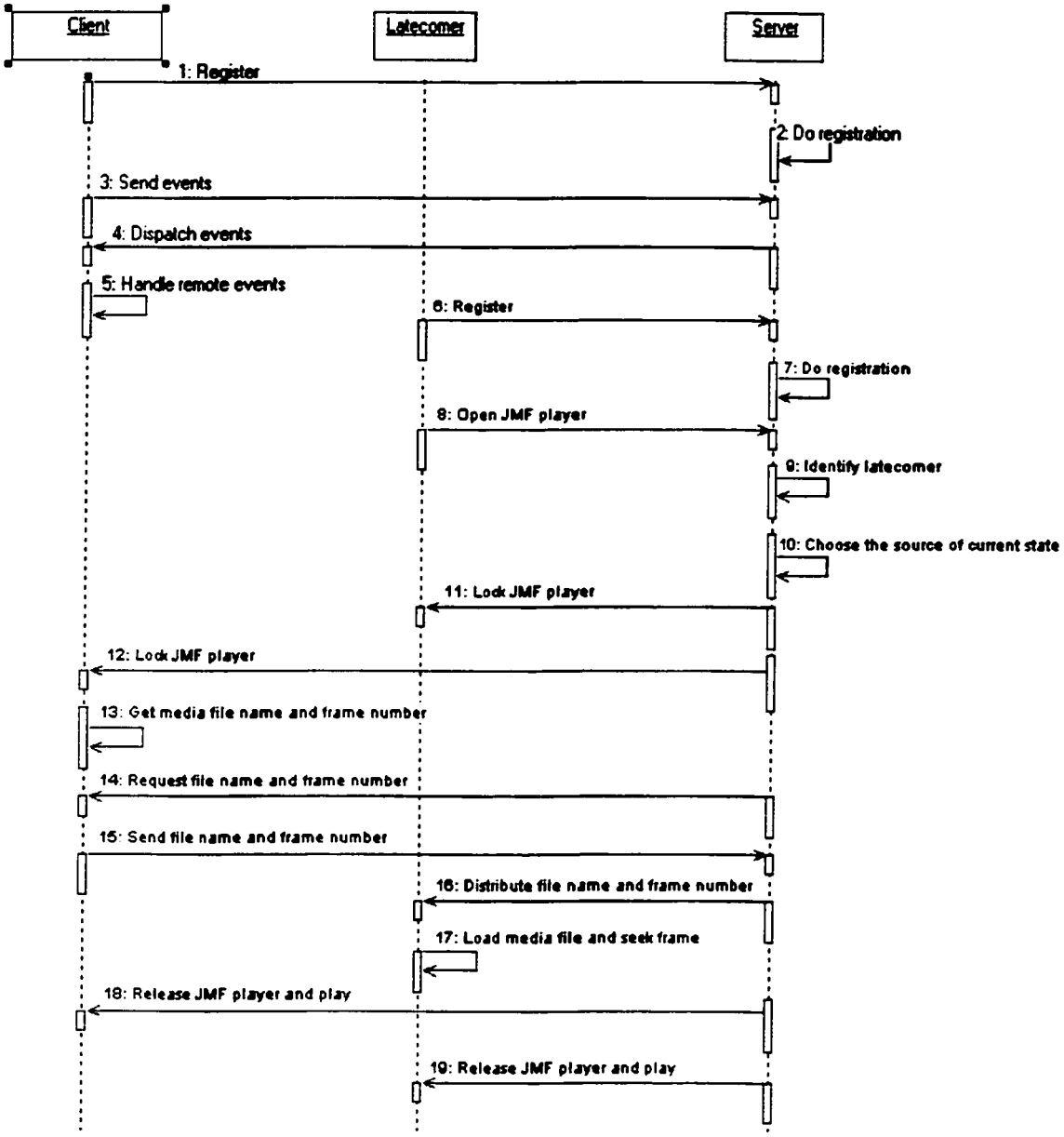

Figure 3.2 The algorithm for the latecomer support for JMF player

Here is the scenario of the latecomer support for the media player: when a client opens the media player, a check message is sent to the server to identify if the client is a latecomer for the media player. If the client is a latecomer, the server dispatches lock messages to both the latecomer and the source of the current state. Once the latecomer receives the lock message, it locks the player and waits for the transmission of the media file name and the frame number. When the source gets the lock message, it stops playing the movie and remembers the frame number where it stopped. Then it locks the player

and sends the frame number and the name of the media file it played to the server. After the server receives the information from the source, it forwards the information to the latecomer. When the latecomer receives the frame number and file name, it releases an unlock message to the server and releases itself too. Then the latecomer loads the file from local storage according to the file name it gets, seeks the frame number and plays the media from the frame it gets from the source. The source starts to play the media file as it catches the release message from the server.

3.1.3 Design of the Session Recorder

Latecomer support lets latecomers get the current state of the shared application. However, sometimes latecomers may also be concerned about what events have happened before they join the session. Therefore, it is necessary to have a session recorder to record all the events happened since the beginning of the session and to replay these events on clients' demands. A session recorder is added to JASMINE and as a result every client (not only latecomers) can replay all the events happened since the session began.

The figure 3.3 shows the algorithm for the session recorder. When the server gets any event from clients, it stores the event in the event log before it sends the event to clients. If a client requires viewing the history, the server sends the event log to the client. Then the client can start another copy of the client software and replay all the events recorded in the event log.

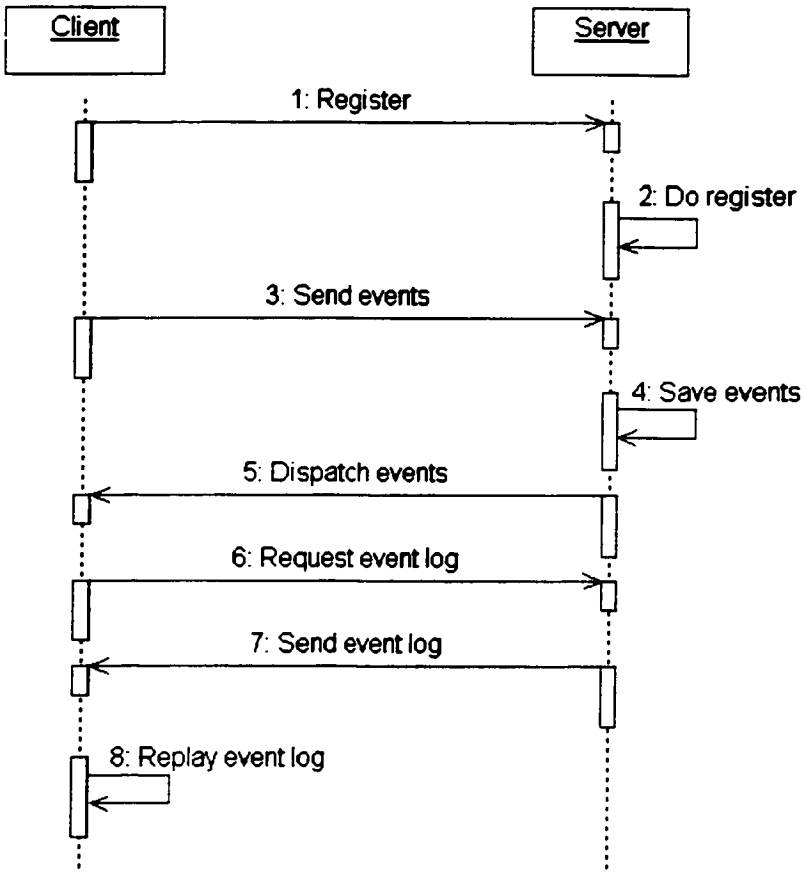

Figure 3.3 The algorithm for the session recorder

As described in the section 2.2.3, JASMINE uses a client-server architecture and all the events occurring on clients must pass through the server. Therefore, in our design, the collaboration server is responsible for recording events. Every client can send a request to the server and require viewing recorded events.

The event log, also called event history, is maintained on the server. Whenever the server receives an event, it records the event into the event history before distributing the event to other clients. When a client wants to view the history, it sends a history request to the server. Then the server passes the event history to the client. After the client receives the event history, it replays the history locally. During the history request and reply procedure, the client does not need to ask for permission to transfer data.

3.1.4 Class Diagrams

3.1.4.1 Class Diagram of the JASMINE Server

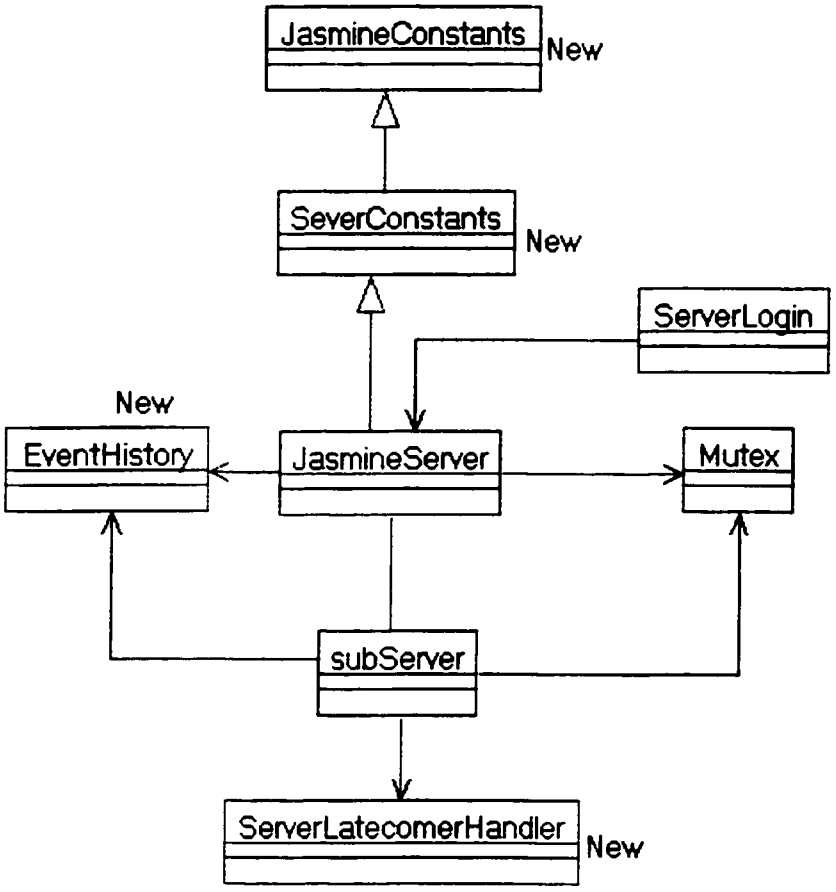

Figure 3.4 Class Diagram of the JASMINE Server

Several new classes are added to the original JASMINE system. The class **EventHistory** is used for recording all the events that change the state of shared applications. Attributes are event and index. Methods are addEvent, getEvent and clearHistory. The class **ServerLatecomerHandler** handles all the messages and operations related to latecomer support, including opening a new socket for the state transmission, remembering all the information related to latecomers, such as states of ongoing-shared applications, latecomer handling states and other relevant information. Important attributes include latecomer, isLatecomer, maxclient, groupname, client ID and opened socket. Important methods are openApplication, checkForApplication, closeApplication,

releaseApplication, openJMFPlayer, releasePlayer, informClient, removeClient and sendHistory. The interface **JasmineConstants** includes all the constants used for both JASMINE server and client software. The interface **ServerConstants** includes constants used only for the JASMINE server software. Classes **JasmineServer** and **subServer** have been modified too.

3.1.4.2 Class Diagram of the JASMINE Client

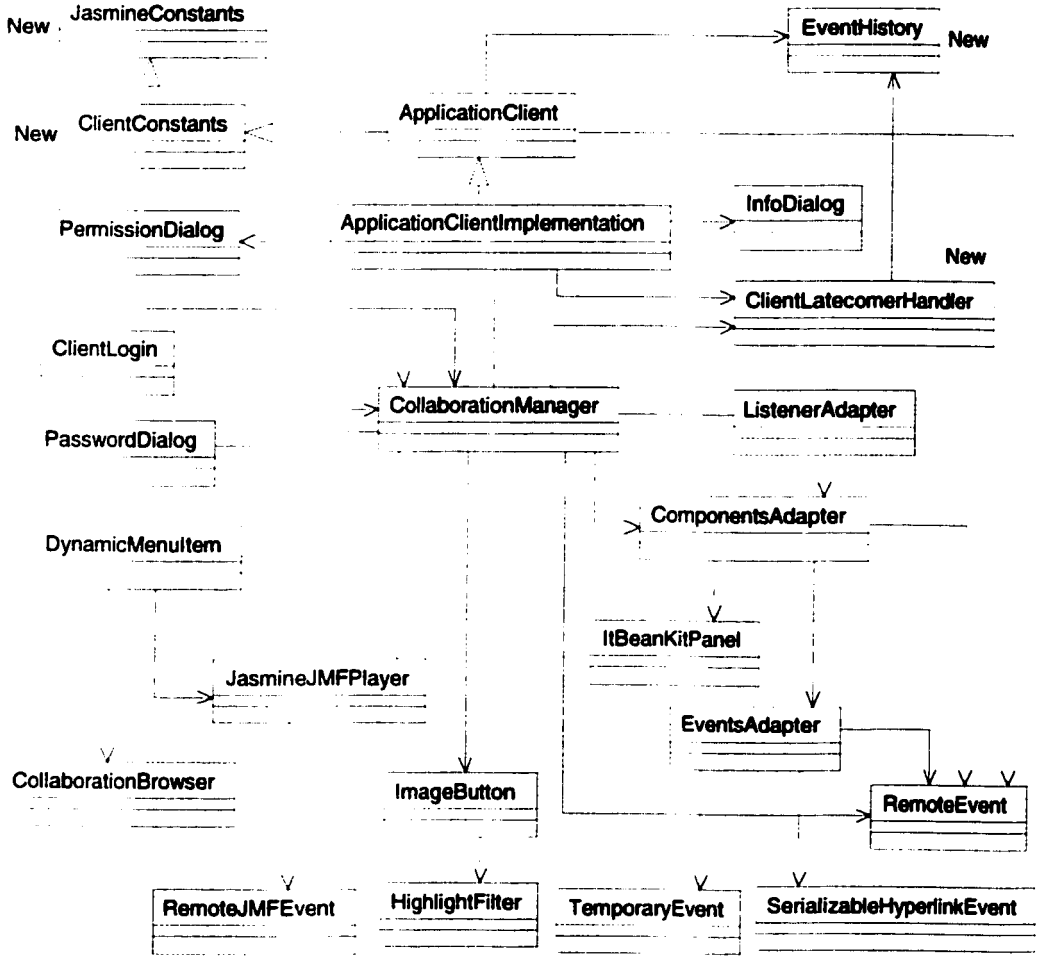

Figure 3.5 Class Diagram of the JASMINE Client

Several new classes are added to the old JASMINE client. The class **ClientLatecomerHandler** is responsible for locking shared applications, remembering components and releasing shared applications. Major attributes are carets and

removedListeners. Important methods are disableApp, enableApp, unconnectListen, connectListener, rememberCarets, setRememberedCarets, disableComponent, enableComponent and getFirstClassMatch. The interface **JasmineConstants** includes all the constants used for both JASMINE server and client software. The interface **ClientConstants** includes constants used only for JASMINE client software. Classes **ApplicationClient, ApplicationClientImplementation, CollaborationManger** and **ClientLogin** have been modified.

3.2 Client Synchronization

3.2.1 Introduction to Client Synchronization

In a network, latency is the time for a data packet to get from one designated point to another. Usually, latency is measured by sending a packet that is returned to the sender and the round-trip time is considered the latency. The latency assumption is that data should be transmitted instantly between one point and another with no delay at all. The factors that contribute to network latency include:

- **Propagation:** This is simply the time for a packet to travel from one point to another at the speed of light.
- **Transmission:** The medium, no matter if it is optical fiber, wireless or other, introduces some delay. The size of the packet also introduces some delay in a round trip since a larger packet will take longer time to be received and returned than a small one.
- **Router and other processing:** Each gateway node takes time to examine and possibly change the header of a packet, for example, changing the hop count in the TTL (time-to-live) field.
- **Other computer and storage delays:** In networks at each end of the journey, a packet may be subject to storage access delays at intermediate devices such as switches and bridges. However, this kind of latency is not considered most of the time.

The tele-collaboration is based on network communication, so network latency is definitely a factor that has impact on the quality of the collaboration. One of the most important objectives of collaboration is to let all the clients share application states and see the same view at the same time. However, due to the different quality of network connections between the server and different clients, it is very common that clients see different stages of the shared application. Latency will cause problems because different clients who see the different views of the shared application may operate differently to the shared objects and cause consistency problems. Here we introduce a network synchronization solution to collaboration systems.

3.2.2 System Design

This solution is to eliminate the latency caused by the network transmission. The basic idea is to synchronize the arrival of events on client sites. Since JASMINE uses a client-server architecture, network transmission latency for different clients cannot be avoided. The objective is to minimize the difference of arriving time of all the clients. Therefore, a mechanism is added to handle the latency caused by the network transmission.

3.2.2.1 Algorithm

The algorithm used in our solution is as follows. The JASMINE server is responsible for handling latency. When a client connects to the server, the server starts to test the transmission time of this connection on a regular basis. The server calculates the latency for every connection based on data it collects from each client. The calculation is re-done after a new detection. When the server receives an incoming event, it distributes the event to clients in the order that are sorted according to their latency. The rule is that the client with the slowest connection has its events sent first. The client with the fastest connection has its events sent last. The interval time between 2 clients is calculated based on the difference of their transmission time.

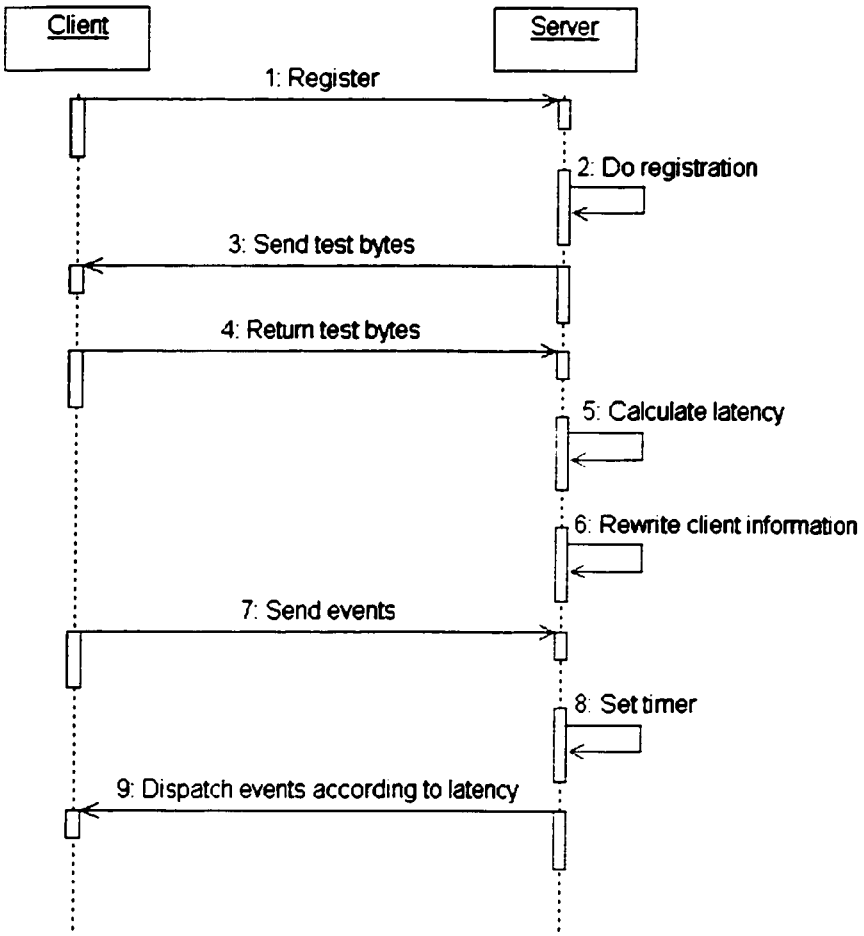

Figure 3.6 The algorithm for synchronization

3.2.2.2 Network Transmission Detection

An echo program is used to detect the network transmission time. Because the JASMINE system is built on top of the TCP protocol, a TCP echo program is used accordingly. The echo program is designed for testing purposes. It sends some testing bytes to the destination machine and sets the time stamp while the process starts. While the testing bytes are sent back, it sets the time stamp again. Using these two time stamps, it is easy to calculate the transmission time of the connection. Since the connection status keeps changing all the time, it is appropriate to test the connection on a regular basis. Users can set the interval time according to the traffic condition of the network.

There are two types of echo systems. The first one is to send testing data to the reserved echo port # 7. In this case, nothing needs to be done at the destination machine. The other type of echo systems needs both the echo client and the echo server to complete the testing. In this case, we need to set an echo server on the destination machine to listen to a specific port. We also need to set an echo client on the other side to send testing data to the echo server.

In our case, we use the second approach. The port 7 is blocked by the system administrator for security reasons, so it is impossible to adopt the first approach. Another reason is that we want this system to be as generic as possible since it is likely that the port 7 is blocked in many places. The echo client is placed on the collaboration server and the server sends testing bytes every 30 seconds (we can set the interval time either shorter or longer). The echo server is set on the client side. Whenever a client is started, it begins to listen to the assigned echo port and waits for testing bytes. If it receives testing bytes, it returns them immediately to the echo client who sent the data. Regarding time stamps, we only record them at the server side. Since JASMINE can be used through the Internet, it is difficult to synchronize clocks of all the clients in the session. Therefore, we record the time testing data are sent and the time the data are received. We use the following equation to calculate the transmission time of one connection.

$T = (T_e - T_s) / 2$

T_s : the start time

T_e : the end time

T : the transmission time

3.2.2.3 Latency Calculation

After collecting the transmission time of all the connected clients, we sort the transmission times in a descending order. The latency of each client is calculated using the following equations:

$L_i = 0$ if $i = 0$

$L_i = T_{i-1} - T_i$

L_i : the latency of client i

i : the id of a client in a sorted array

When the server receives an incoming event, it distributes the event to other clients according to its latency calculated before. First, the server sends the event immediately to the one that has the latency value of zero. Then it waits for a period of time, which equals the latency of the second client before it sends the event to the next client. The server repeats the procedure until the event has been sent to all the clients in the session.

3.2.2.4 Class Diagram

1. Class Diagram for the JASMINE Server

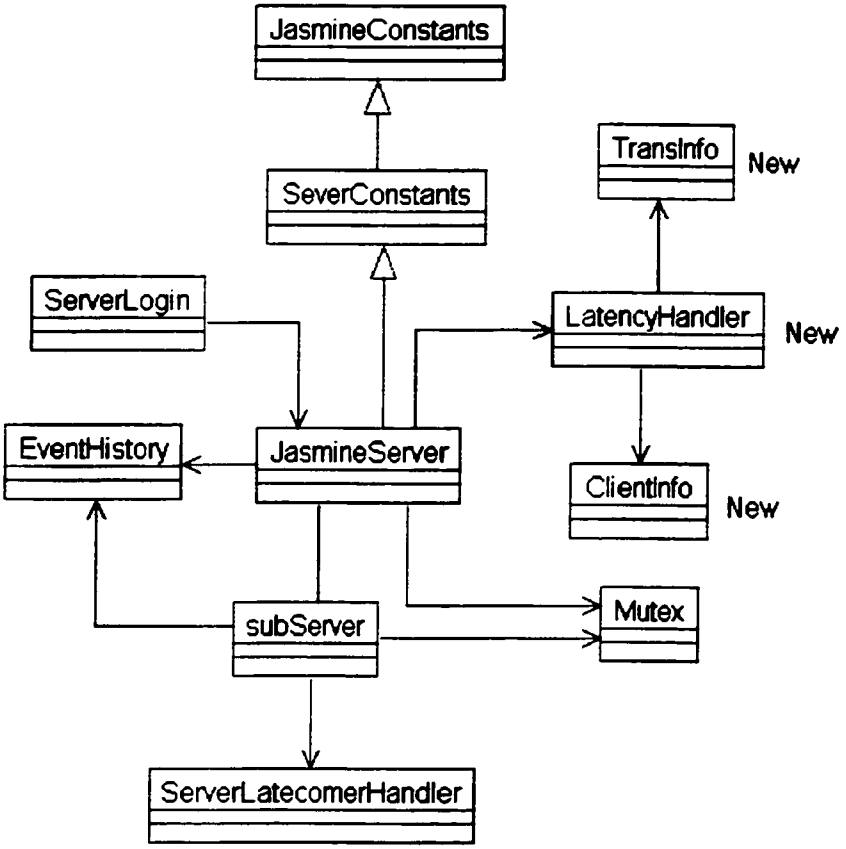

Figure 3.7 Class diagram for the JASMINE server

Some new classes are added. The class **ClientInfo** stores all the information of clients, including client ID, host name, user name, transmission time and latency. major methods are to get and set different information. The class **LatencyHandler** tests network connections, calculates latency and stores the information to the class ClientInfo. Major attributes include client information and max client number. Methods are getLatency, getTransTime and sortClient. The class **TransInfo** opens a window to show some client information related to network connection. Major attributes are transmission time, client ID and name. Class **ServerLogin** and **subServer** have been modified too.

2. Class Diagram for the JASMINE Client

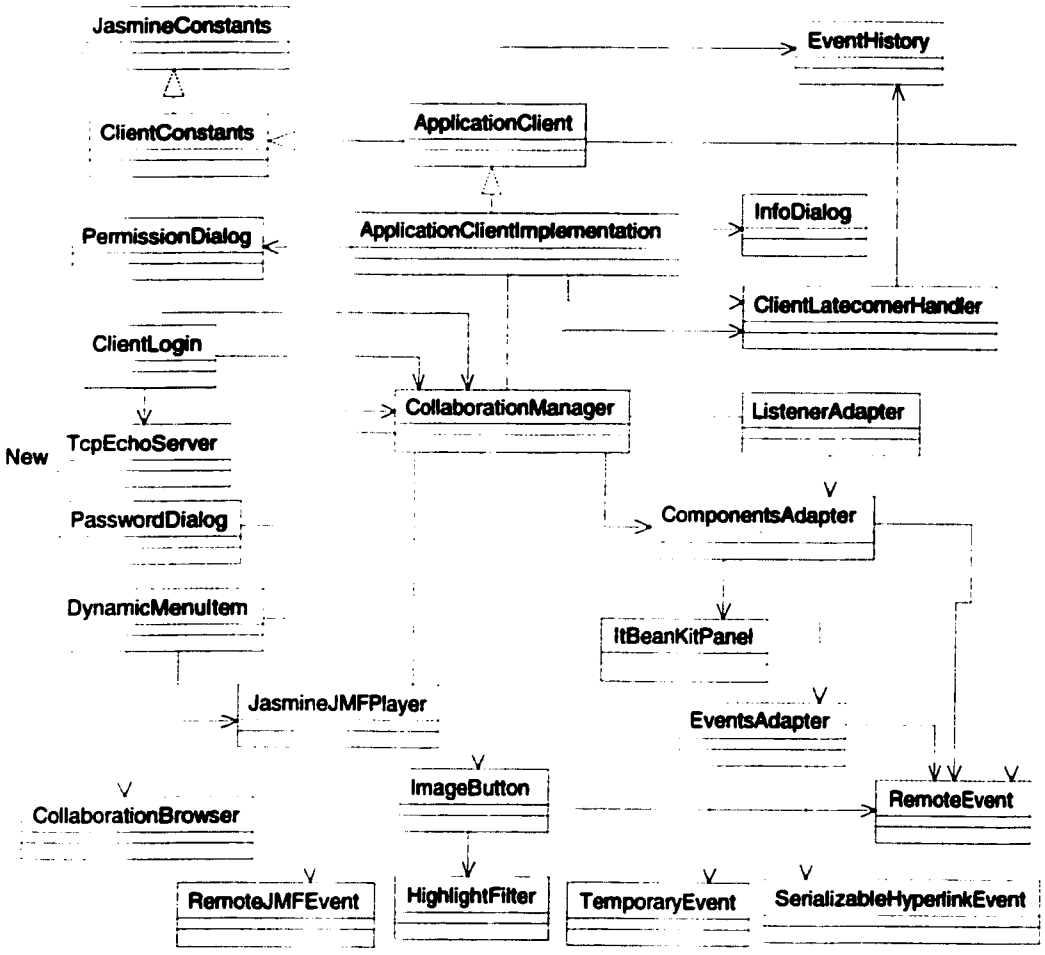

Figure 3.8 Class diagram for the JASMINE client

The class **TcpEchoServer** is used to send test messages back to the JASMINE server. It is used to test network connections and calculate latency. The attribute of the class is port. The major method is "go". Classes **ClientLogin, ApplicationClient, ApplicationClientImplementation** and **CollaborationManger** have been modified.

3.3 Multi-session Support

3.3.1 Introduction to Multi-session Support

The objective of synchronous collaboration is to provide a shared workspace to users and allow them to work collaboratively like in a real meeting room. Sometimes, people would like to work in different groups for different tasks. Therefore, they may want to work in different shared workspaces. A multi-session collaboration system presents users with such an environment.

Multi-session support has been implemented in many projects and works well. In these systems, centralized servers are used to control the session management. As we mentioned in the related work (see section 2.1), the session management is integrated firmly into these systems and it cannot be removed or replaced. However, if we use a distributed architecture, such as a peer-to-peer network, this type of multi-session implementations are not suitable any more.

In our project, a lightweight multi-session management has been designed and implemented. It is implemented in a client-server architecture, but it can be easily migrated to a distributed system.

3.3.2 System Design

The multi-session support in JASMINE is a lightweight service to the system. It allows clients to create a new session or join an existing session. It also provides the function that lets clients check the session information, such as the list of active sessions and the members of each session. The session information changes dynamically and clients can always view the updated session information.

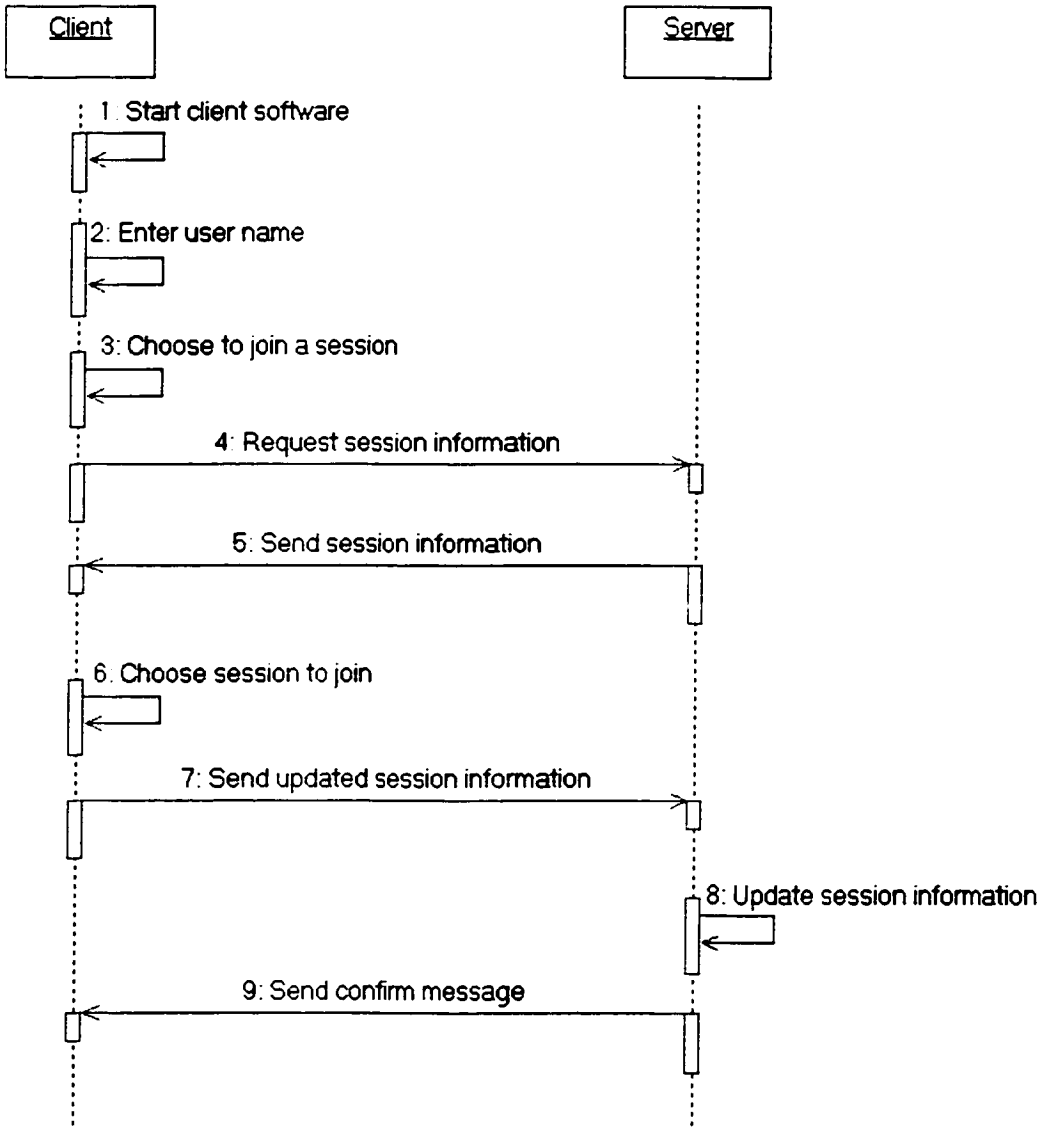

Figure 3.9 The message flow of joining a session

Figure 5.1 shows the message flow of the operation of joining an existing session. Whenever clients start the client software, they have to choose which session to join or to create a new session before they can do any other operation. If a client starts a new session, the session will be started with the default state. The system does not remember the state the on which client worked last time. If the client joins an existing session as a latecomer, the system transfers the current state of shared applications to the client.

The session information is stored on the server. However, it is not separated from other client information. It is a part of the client information. The server checks every event or message sent to it for the session information to make sure that the event or message will be distributed to clients in the same session as the sender. The session references are added to most stored data, including connection data, latency data and latecomer data.

All the services in the original system are modified in order to work in the multi-session environment, including latecomer support and client synchronization. Since session references are added to all the data related to these services, it is easy to give the right service to the right client in the right session.

Compared to other session management systems, this session management supports only simple session control, but not the complete room control. The reason is to simplify the server's functionality. As we mentioned before, the client-server architecture of JASMINE has a critical problem, scalability. In order to solve this problem, we are trying to migrate the JASMINE to a peer-to-peer architecture. Because there is no central control server in a P2P system, all the information has to be stored on each client. A lightweight session management service makes the migration to P2P more easily and reliably since all the session information has already integrated into the client information. What to do next is to move the client information from the server to clients.

3.4 Collaborative VRML Viewer

VRML viewers are used to display VRML files. By adding VRML plug-ins or some additional software, many popular browsers, such as Microsoft Internet Explorer and Netscape Navigator, can be used as VRML viewers. Collaborative VRML viewers can be used as not only VRML viewers but also collaborative tools that let users work in the VRML 3D world together.

3.4.1 The Reason to Choose blaxxun3D

Besides Blaxxun3D, there are still some other products supporting VRML. However, most of them need to be installed on clients' machines. One of the most important features of JETS is that it does not need to install any additional software on clients' machines. It would be appropriate to keep this feature for the VRML viewer. Using Blaxxun3D, clients do not need to install any software or plug-in to view VRML files. This is the most important reason why we chose Blaxxun3D as out platform to support VRML. It also has some other advantages [27].

- Compact standard Java-applet
- No plug-in and therefore no extra download or installation required
- Simple integration with existing home page
- Platform-independent
- Compatible with VRML
- Supports 3D objects and interactivity
- Extensible with built-in API interfaces

However, blaxxun3d has also some limits [27].

- Blaxxun3D supports a VRML subset described in the specification. The content can only use the featured subset. VRML code must be cleaned up before using.
- Blaxxun3D offers no built in navigation control. (You have several ways to navigate through worlds displayed with blaxxun3D's: 1. Navigating with binding of several different viewpoints; 2. Walk mode / examine mode.)
- Blzxxun3D doesn't support VRML Script-Nodes.
- Blaxxun3D only supports ".au" files sampled at 8000hz.
- Blaxxun3D support the GIF as image Texture.
- Blaxxun3D doesn't support viewpoint animation through setting viewpoints as children nodes of animated transform nodes.
- The speed is slow because of using Java.

3.4.2 System Design

3.4.2.1 Algorithm for Collaborative VRML Viewer

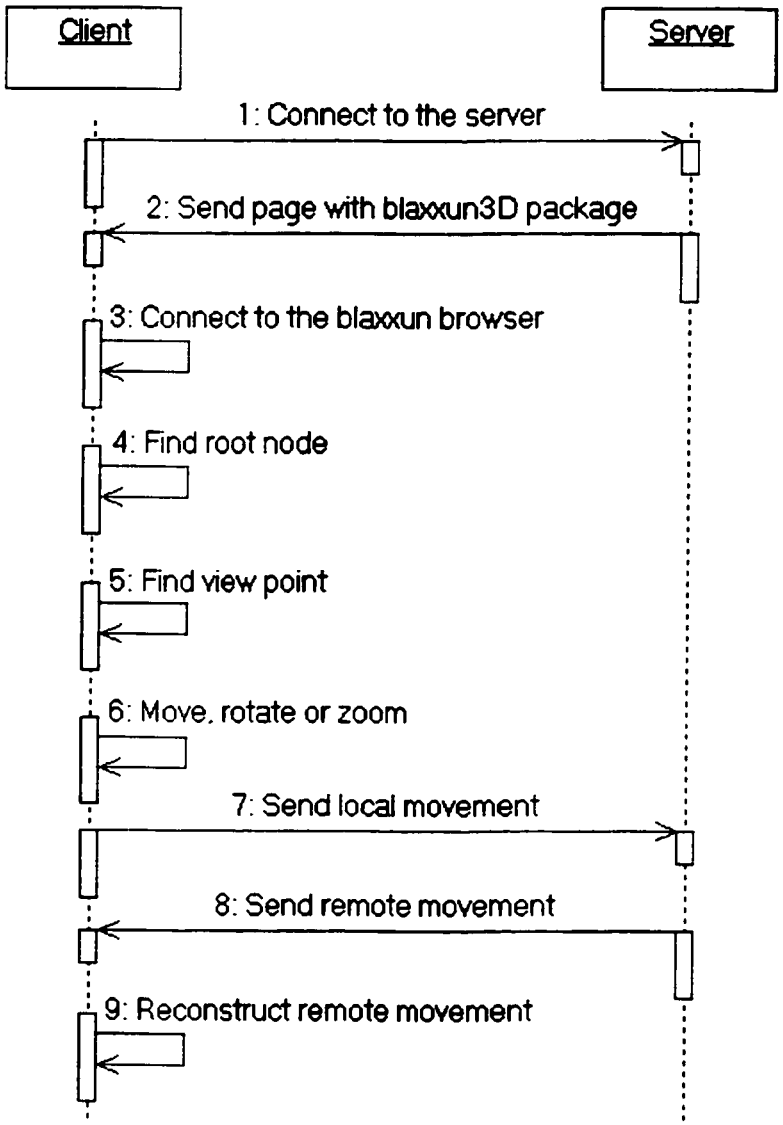

Figure 3.10 The algorithm for collaborative VRML viewer

3.4.2.2 Modifications to the Existing System

1. Modification to the existing VRML server

The existing JETS VRML server already has abilities to receive incoming messages from clients and distribute messages to clients. Therefore, we made only minor modifications in order to match modifications made to the client software.

2. Modification to the existing VRML client

The existing VRML client has its own VRML parser that can only show the wire frame of 3D objects, so this whole part is discarded and Blaxxun3D is used instead. The existing data transmission part is reused.

3. Connecting with Blaxxun3D browser

Blaxxun Corporation provides an API for further development, such as navigation implementation. The company also provides a standard connection method for Java applets. The method is used to verify if a Blaxxun browser is started.

4. 3D world navigation

One of the most important parts of VRML viewers is to interact with objects in the 3D world. Blaxxun3D provides an API for operation on objects. In order to navigate in the 3D world, first the browser needs to find the root of the world. Then it should get viewpoints of the world. Finally, it must get nodes according to mouse movements. The movements, including zooming, rotating and moving, are programmed using existing standard algorithms.

3.5 The New Architecture of JASMINE

After the development of latecomer support, client synchronization and multi-session support, the JASMINE structure has been changed. For the JASMINE server, the original one has only a communication module, which is used to communicate with clients and to

provide floor control and moderation. Now the new server has some new features, such as latecomer handler, latency handler, session controller and session recorder. These new functions make the JASMINE server a powerful collaboration server. The new architecture of the JASMINE server is shown as follows.

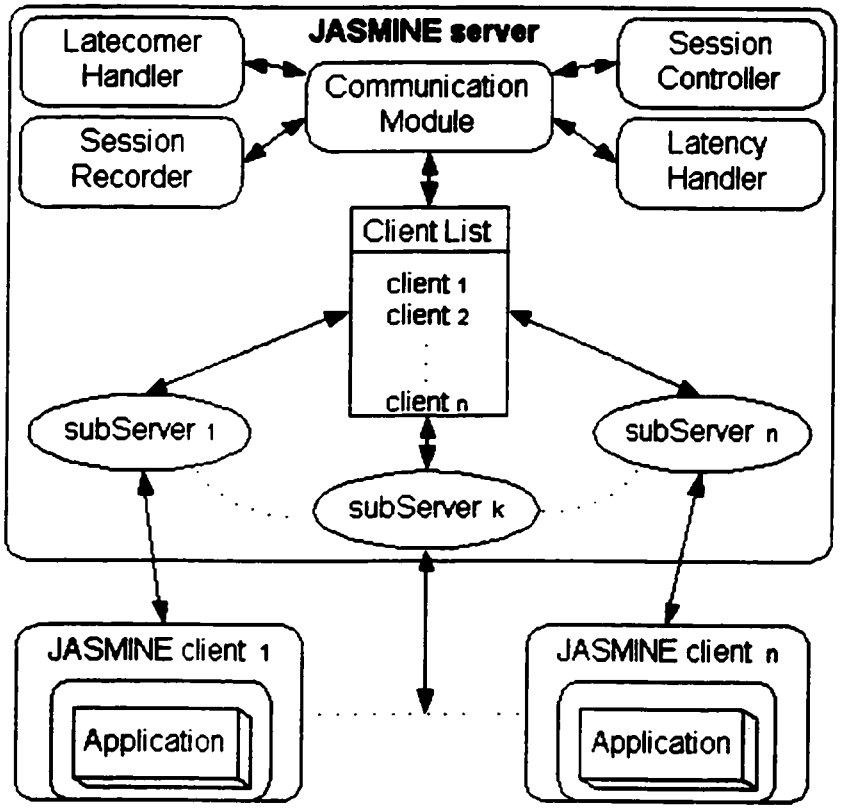

Figure 3.11 The new structure of the JASMINE server

The **Communication Module** is responsible for the communications with JASMINE clients. It keeps listening to opened sockets, receives messages from clients and sends messages to clients. The floor control and moderation is also handled in this module. The **Latecomer Handler** handles all the messages related to latecomers' requests and responses. It handles the transmission of the current states as well. The **Session Recorder** records all the events that change the state of shared applications and sends these recorded events to clients who require viewing them. The **Latency Handler** tests network connections in a regular basis and calculates latency for each client. The communication module uses the calculated latency when it distributes events to clients.

The **Session Controller** lets users create a new session, join an existing session or leave a session.

For the JASMINE client, besides the original event adapter, component adapter, listener adapter and collaboration manager, latecomer handler, latency controller and session controller are added. Some of these new features are optional, like latency controller. Others are mandatory, like the latecomer handler and session controller. These new features give clients more choices for collaborative work. The following figure shows the new structure of the JASMINE client.

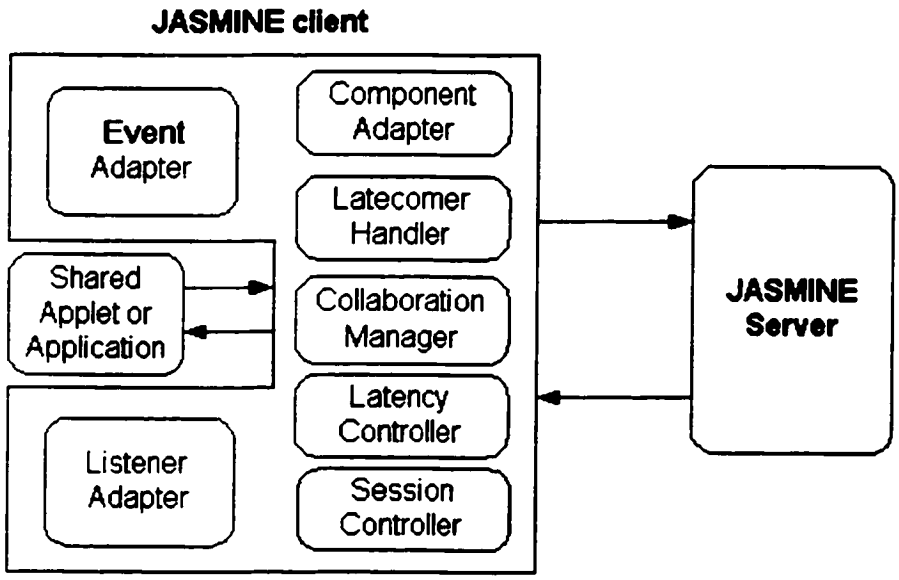

Figure 3.12 The new structure of the JASMINE client

Besides the components in the original JASMINE system, some new components are added for new functionality. The **Latecomer Handler** is responsible for locking shared applications, recording components and releasing shared applications. The **Latency Handler** is used for network transmission test. It sends test messages back to the JASMINE server. The **Session Controller** maintains all the information related to the session that users work with.

Chapter 4 Implementation

4.1 Latecomer Support

Like the system design chapter, this section is also divided into two parts. One part is on the latecomer support for Java applications, Java applets and the JMF player. The other part is on the implementation of the session recorder.

4.1.1 Implementation of Latecomer Support

The system is programmed in Java following the design we mentioned in the former design chapter. However, there are some key points that need to be explained.

1. *The source of the current state*

If the ongoing session is a moderated session, the server chooses the moderator to be the source of the current state. Otherwise, the server chooses the one who starts the session to be the source of the current state. If the client who started the session has left the session, the server chooses the client who has stayed in the session for the longest time as the

source of the current state. The source of the current state is chosen in the same way for Java applications, Java applets and the JMF player.

2. *Transferring application states*

In order to transfer current states of Java applications and applets, the system opens new sockets only for state transmission. On the client side, an ObjectInputStream and an ObjectOutputStream are used for state transfer through networks. However, when clients read objects from I/O streams, objects need to be instantiated. Therefore, all the Java applications and applets that need to be transferred have to be serializable. Otherwise, the system will throw ClassNotFound exceptions. On the server side, the server opens a BufferedInputStream and a BufferedOutputStream for state transfer. The advantage of using these two streams is that the server can read bytes from a stream without necessarily causing a call to the underlying system for each byte read. Therefore, the server does not need to have all the classes been transferred. The date is ready by blocks into a buffer, then the server can read from the buffer of the BufferedInputStream and write bytes to the buffer of the BufferedOutputStream immediately.

For transferring the JMF player current state, there is no need for a new socket. The system uses the already opened socket to transfer the media file name and the current frame number.

3. *Lock process*

The lock process is necessary for preventing the source of the current state and the latecomer from doing any operation while they are transferring states. The lock process for Java applications and applets is similar to the work done by the team at the Ulm University [14]. Before sending the current state of shared applications to the server, clients must lock shared applications. This is to avoid further operation on shared applications during the state transmission. The lock process has four steps. First, all the listeners are disconnected from components. Second, all the components of the shared

application are disabled too. Third, the system remembers carets of the shared application. Finally, the system remembers the Look and Feel of the shared application. The system stays in this status until it gets the unlock message from the server.

The lock process for the JMF player differs from the lock process for Java applications and applets because their work mechanisms are different. On the latecomer side, the client removes the player controller from the player and waits for the transmission of the media file name and the frame number. On the source side, the client stops the player and gets the frame number where it stops. Then it removes the player controller from the player and sends the media file name and the frame number to the server. Then the client waits for the release message from the server.

4. *Release process*

For Java applications and applets, the release process is like the reverse process of the lock process. First, the client system sets the LookandFeel of the shared application. Second, the system sets the carets of the shared application according to what it remembered in the lock process. Third, the system enables all the components. Finally, all the listeners are connected to their components.

For the JMF player, the release process works differently from the one for Java applications and applets. On the source side, when the client receives the release player message, it adds the player controller to the player, seeks the frame number, and starts playing from where it stopped. On the latecomer side, when the client receives the file name and frame number, it adds the player controller back to the player, loads the media file according to the file name it receives, seeks the frame which matches the frame number it has got, and starts playing from this frame.

4.1.2 Implementation of the Session Recorder

The event log, also called the history, is maintained on the server side. An EventHistory class is used to store all the coming events. Only events that change states of the shared applications are recorded. Other signals, like floor control signals and moderation signals are not written into the event history. If a client wants to view the history, he/she sends a history request to the server. Upon receiving the request, the server sends a vector that contains all the saved events to the client. After the client gets the event history, it starts another copy of client software and replays all the events in the event history.

The playing of the session recorder is not collaborative. Only the client who requires the history is able to view the history. The floor control is not functioning for the session recorder. Even if some clients are operating on shared applications, others can still play recorded events at the same time on their own machines.

4.2 Client Synchronization

In order to implement the algorithm we introduced in the system design chapter, the class LatencyHandler is placed on the server side to handle operations related to latency operations. The latency handler includes a method to detect all the network connections from clients and get the transmission time for every client, a method to calculate the latency of each client who is already in the session, and a method to write all the information to a class, which contains all the information of clients. The connection detection is done by setting an echo client and several servers and testing the round trip time of test packets. The latency calculation is based on the algorithm we mentioned in the system design chapter. All the connection information is recorded in the ClientInfo class with other client information. Other than these, a window to display a table of certain information about all the network connections, including host names of clients and transmission times both in digital format and in a progress bar. From this table, users can easily get the information of all the network connections. The digital data format allows users to know the exact transmission time. The progress bar allows users to easily compare different connections.

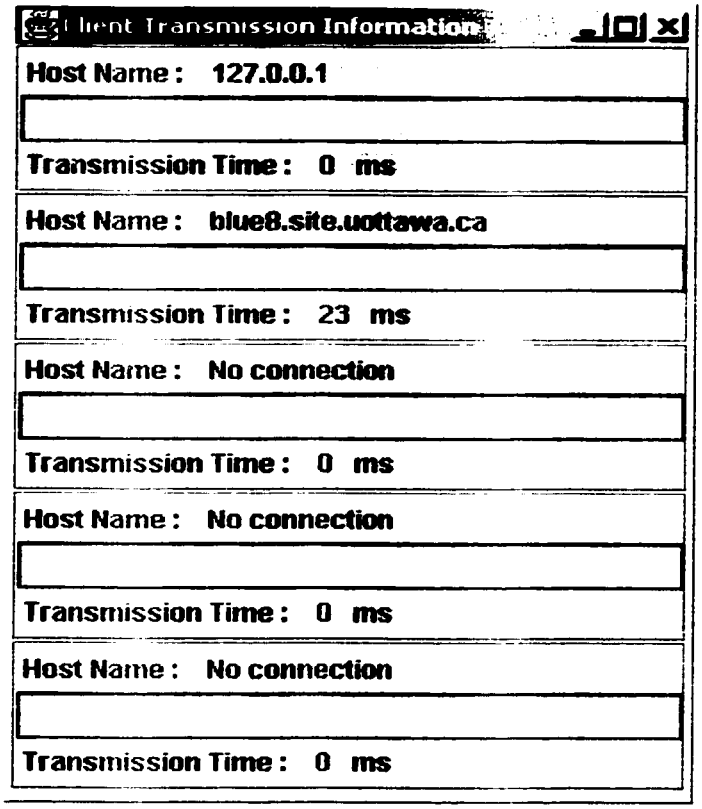

Figure 4.1 The interface of client transmission information on the server

The echo client is placed on the JASMINE server. It sends test packets to all the connected clients in a regular basis. Echo servers are set on clients' sites to listen to connections to the JASMINE server. The echo server is started when a client starts the system and it keeps listening to the assigned echo port during client's lifetime in a session. The echo server is optional to clients. When users start the client software, they have the choice either to start the echo server or not.

An important issue in the implementation is the size of test bytes. If the size of test bytes is bigger, it takes more time to receive them and return them to the sender. Since our objective is to synchronize the arrival of events, we choose the size of 16 bytes, which is similar to the average size of different events. Since the size of the packet is small, we ignore the processing time of receiving and sending on end points and points on the route.

4.3 Multi-session Support

The session management is located on the collaboration server. A client information class is used to store all the information of connected clients. The session information is also stored in this class. When the server gets an event from a client, it checks the client information first and finds out which session the client is in. Then the server checks all the connected clients to find out all the clients in the same session as the sender and distributes the event to these clients.

In the client interface, users have three choices, create, join and display. The choice "Create" is used to create a new session. In the interface of creating a new session, users can enter the name of the session and create the session after pressing the OK button. The choice "Join" is for users to add themselves to an existing session. Users get a list of all the existing sessions and they can choose one to join. The choice "Display" is used to display the session information. Users can choose one session from the session list and display all the clients in this session. Users may check the session information before they make the decision whether to create a new session or to join an on going session.

For services like latecomer support, session recorder and client synchronization, the session management becomes more difficult since the information related to these services is more complicated. In order to solve the problem, the information related to latecomers and latency is stored with session references. When the server needs to handle latecomers, it checks session references first and sends requests only to clients in the same session as the sender for the transmission of the current state. The session recorder records events in every session in a separated space. It a client in a session requires viewing the history, the server sends the history of this session to the client. For client synchronization, the server tests all the connections no matter which sessions they are in. The connection information is stored in the ClientInfo class with session references. When the server receives an event, the server checks the client information and verifies which session the sender belongs and distributes the event only to clients in the same session as the sender.

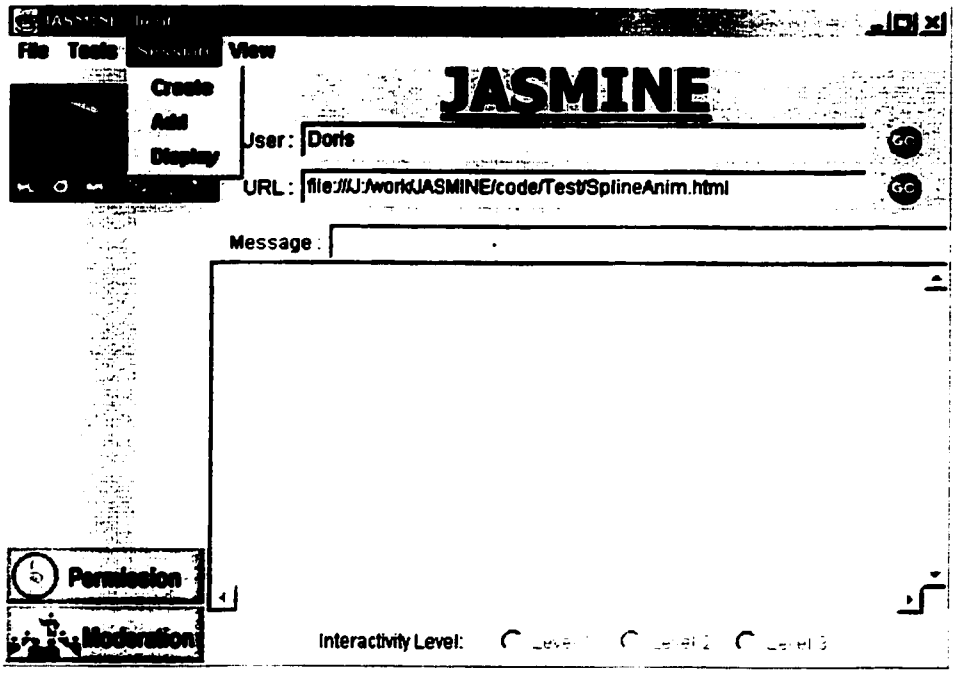

Figure 4.2 The client interface after adding multi-session support

Since the session information is stored with other client information, it is not necessary to have a separate session manager to handle the session information. The session information can be updated and retrieved dynamically. The client information, which includes the session information, can be stored either on the server or on clients' sites. If a centralized architecture is adopted, the client information can be stored on the server. If a distributed architecture is used, the client information can be stored on clients. In this case, before communicating with any one, the sender can retrieve the session information from the client information and send events or messages only to people in the same session.

4.4 Collaborative VRML Viewer

The collaborative VRML viewer is a Java applet. Users can use it to view VRML files and navigating in the 3D worlds collaboratively by connecting to the collaboration server. The Blaxxun3D package is downloaded from the server to clients when clients load web pages on the server. This procedure is transparent to clients. The size of Blaxxun3D

package is very small and it takes very short time to load it into clients' systems. The Blaxxun3D package has to be placed on the root directory of the web server.

To the existing server, a filter is added to block unwanted data from clients because Blaxxun3D has it own data transmission mechanism that cannot be used in our system.

To the existing client, some functions are added to collaborate with the Blaxuun3D parser.

- Create connection with Blaxxun3D browser in order to use its parser to display a VRML world.
- Add navigation abilities to the VRML world. Since Blaxxun3D does not provide any navigation, we had to implement 3D world navigation using Java and Java script.
- Add event listener to the Blaxxun3D world to catch events happened to 3D objects and to get information needed.
- Integrate the parser and navigator to the message transferring mechanism and send messages to the JETS VRML server.
- Add functions to move 3D objects according to the messages clients get from the server.

The user interface has also been changed. The new interface comes with Blaxxun3D and it can be enhanced by adding customized Java script programs.

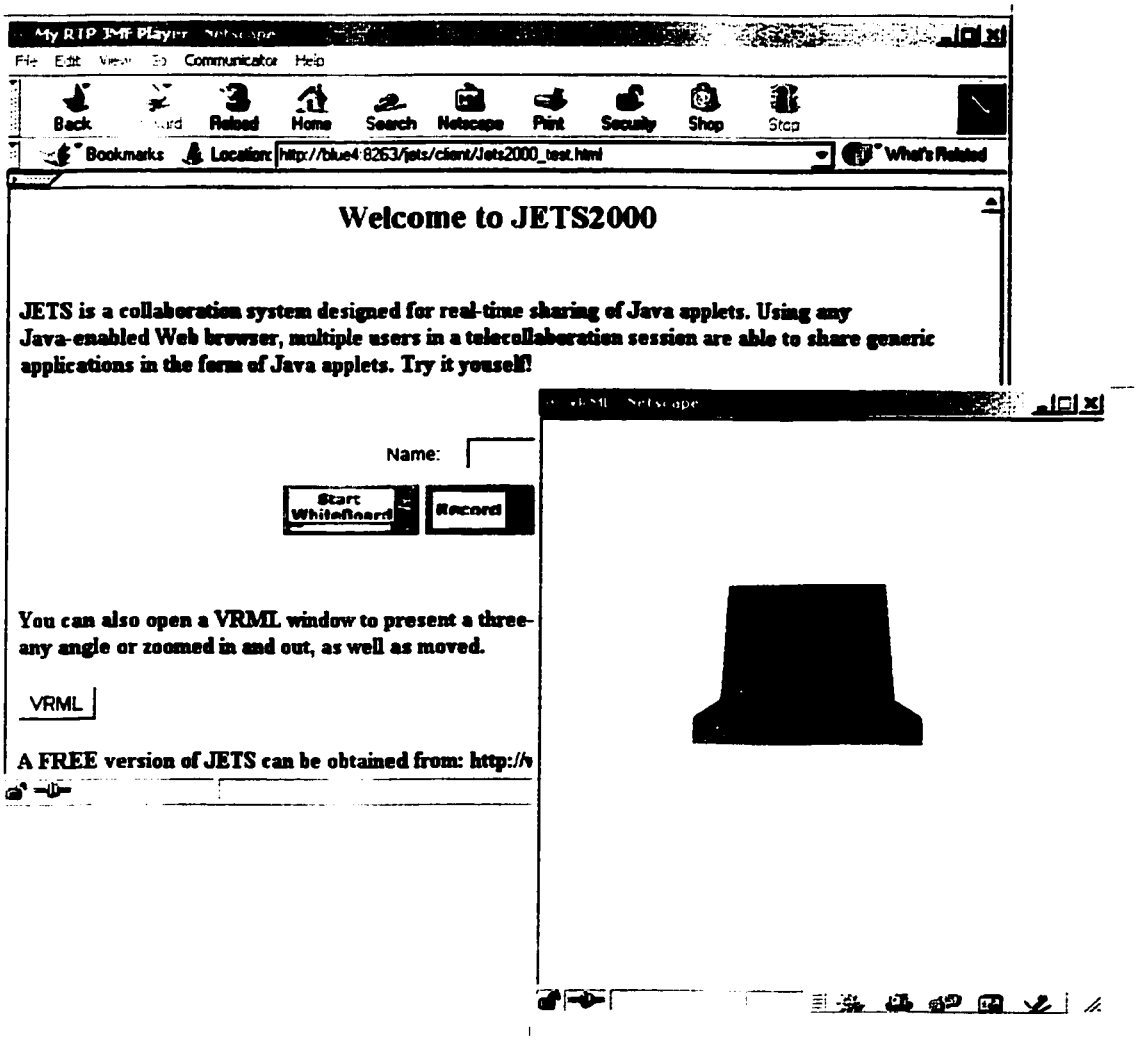

Figure 4.3 The interface of VRML viewer in JETS

Chapter 5 Evaluation

We have done some evaluations to the JASMINE system. Evaluations are in three categories, performance, reliability and usability. We have done some experiments on the performance of the latecomer support and the usability evaluation for the whole system. Other evaluations are based on users' experience.

5.1 Performance

5.1.1 Latecomer Support

After adding the latecomer support service, the performance of JASMINE changes depending on the number of latecomers and the size of shared applications in progress. Usually, it takes about 1 second to update a Java applet or application. But it takes about 6 – 10 seconds to update a JMF player. The update time is the duration from the time that a shared application gets locked to the time that the shared application gets released.

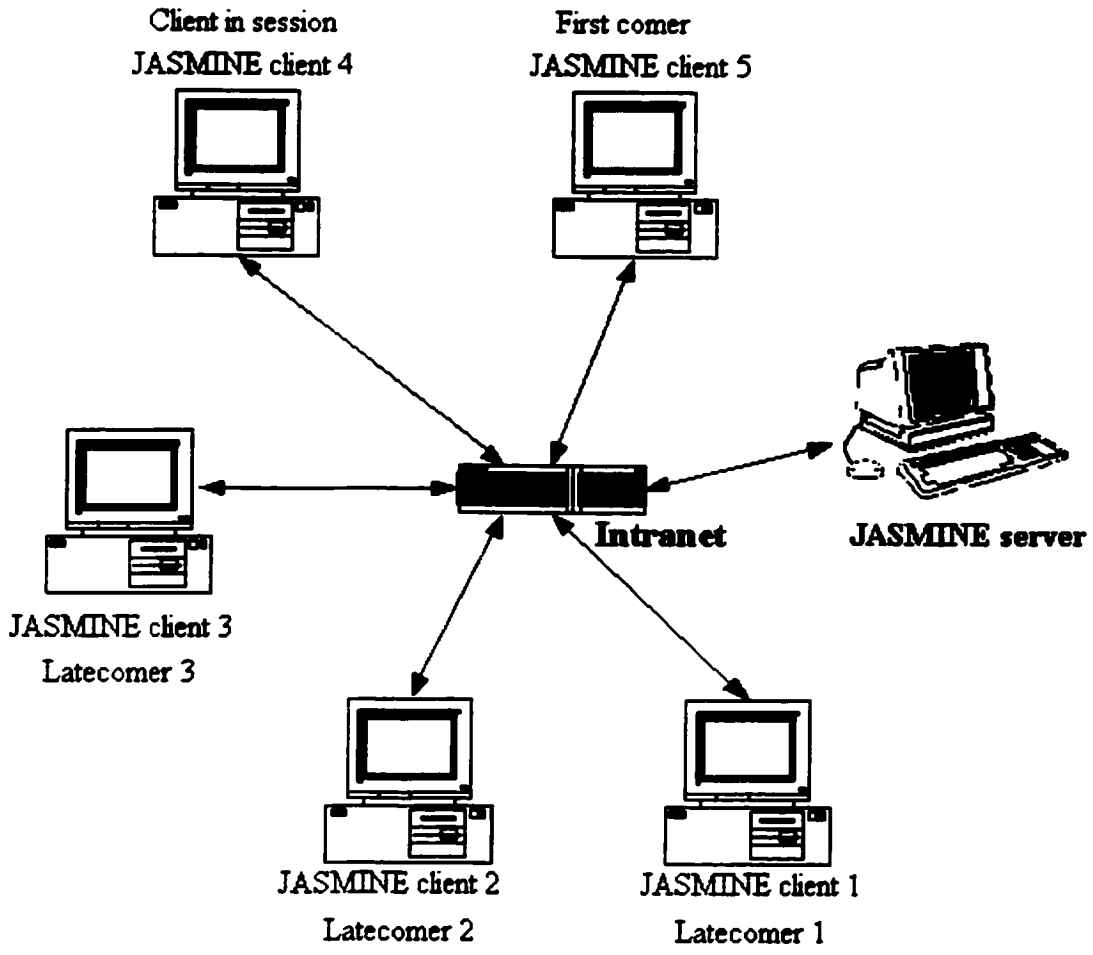

Figure 5.1 The network setting for performance evaluation

We conducted an experiment to test latecomer update time for different users through different connections. As displayed in the figure above, five clients are connected to the JASMINE server. Client 5 is the first one who starts the session. Client 4 is already in the session when latecomers come. The others are latecomers. All the clients are in the same intranet as the server. Among these latecomers, latecomer 1 is a 333 MHz machine and latecomer 2 and 3 are 800 MHz machines. We tested three different size Java applets and a JMF player. The experiment has been done for several times and average values are calculated based on data collected from all the experiments. The result (update time) is the duration from the time to lock a shared application to the time to release the application. The result of the measurements is as follows.

Latecomer	Java Applet 1 (Size: 4K)	Java Applet 2 (Size: 8K)	Java Applet 3 (Size: 24K)	JMF Player
Latecomer 1	1,842 ms	1,963 ms	1,502 ms	6,069 ms
Latecomer 2	1,875 ms	2,016 ms	1,438 ms	6,125 ms
Latecomer 3	1,865 ms	1,988 ms	1,433 ms	6,112 ms
* Latecomer 1	2,233 ms	2,353 ms	1,903 ms	6,108 ms
& Latecomer 2	1,985 ms	2,016 ms	1,640 ms	6,128 ms
** Latecomer 1	1,996 ms	2,153 ms	1,785 ms	6,225 ms
& Latecomer 2	2,059 ms	1,986 ms	1,806 ms	6,117 ms
& Latecomer 3	1,940 ms	2,023 ms	1,752 ms	6,106 ms

* Two latecomers require the current state at the same time.

** Three latecomers require the current state at the same time.

Figure 5.2 The update time for different users through different connections

As listed in the figure 5.2, the result shows that the average update time for various applications is less than 10 seconds. Since the test is done in an intranet, the transmission time is almost zero. If clients connect to the server from other places through other connections, the transmission time should be considered and theoretically, the update time for these clients depends on the connection quality. The update time has little relation with the size of Java applets or the speed of latecomers' computers. However, the number of objects that are in the scene has a little negative impact on the update time. The size of the applet 2 is smaller than the size of the applet 3, but the number of objects in the applet 2 is more than that of the applet 3. We can see from the result that the update time for the applet 2 is longer than that of the applet 3. The number of clients that require the current state at the same does not have much impact on the update time either. In a small or medium group, all the latecomers can get the current state from server at almost the same time.

One major problem with this solution is the collaboration quality of the moderator or the client who starts the session first. The moderator and the starter are the source of the current state, so they are responsible for providing current states to any latecomer who

registers to the session. Since the operation of providing the current state involves locking shared applications, the moderator and the starter will find that their work is often interrupted by latecomers. A possible solution is choosing the source of the current states randomly or in a specific order.

Scalability is another problem with the system. Since the system is built on a client-server architecture, it is difficult to handle the scalability problem. Now we are working on migrating the whole system to a peer-to-peer network and we believe that this will solve the scalability problem.

5.1.2 Client Synchronization

In order to implement the client synchronization service, some components are added to the JASMINE system. On the client side, an echo server is added to test the network connection to the server and it keeps listening to a specific echo port. When the echo server receives test packets, it sends the packets to the sender immediately. Compared to the network transmission time, the local processing time is usually ignored. Therefore, the client synchronization service has little impact on the performance of clients.

On the server side, the server starts with the latency handler, which is responsible for testing network connections and calculating latency. The interval time to test network connections is the key factor that may hurt the performance of the whole system. If users choose to test network connections in a very short interval time, the system performance will be slowed down definitely because the network traffic becomes heavy. When we did the usability evaluation, we let the server test network connections every 10 second. It is obviously that the responding time has been slowed down. However, if users choose the interval time carefully, the client synchronization service does not become a heavy workload to the server.

5.1.3 Multi-session Support

In order to add multi-session support, the storage format of client information is changed. The server needs to do more checking and calculation every time when it receives events or messages from clients. This means that it takes more time for server to distribute events or messages to clients. Since JASMINE uses the client-server architecture that is suitable for only small to medium group of users, the time used for the session management is tolerable. In our experiment, five users connected to the server and they belonged to three different sessions. The responding time had not changed obviously. We will test the system performance in a larger network in the future.

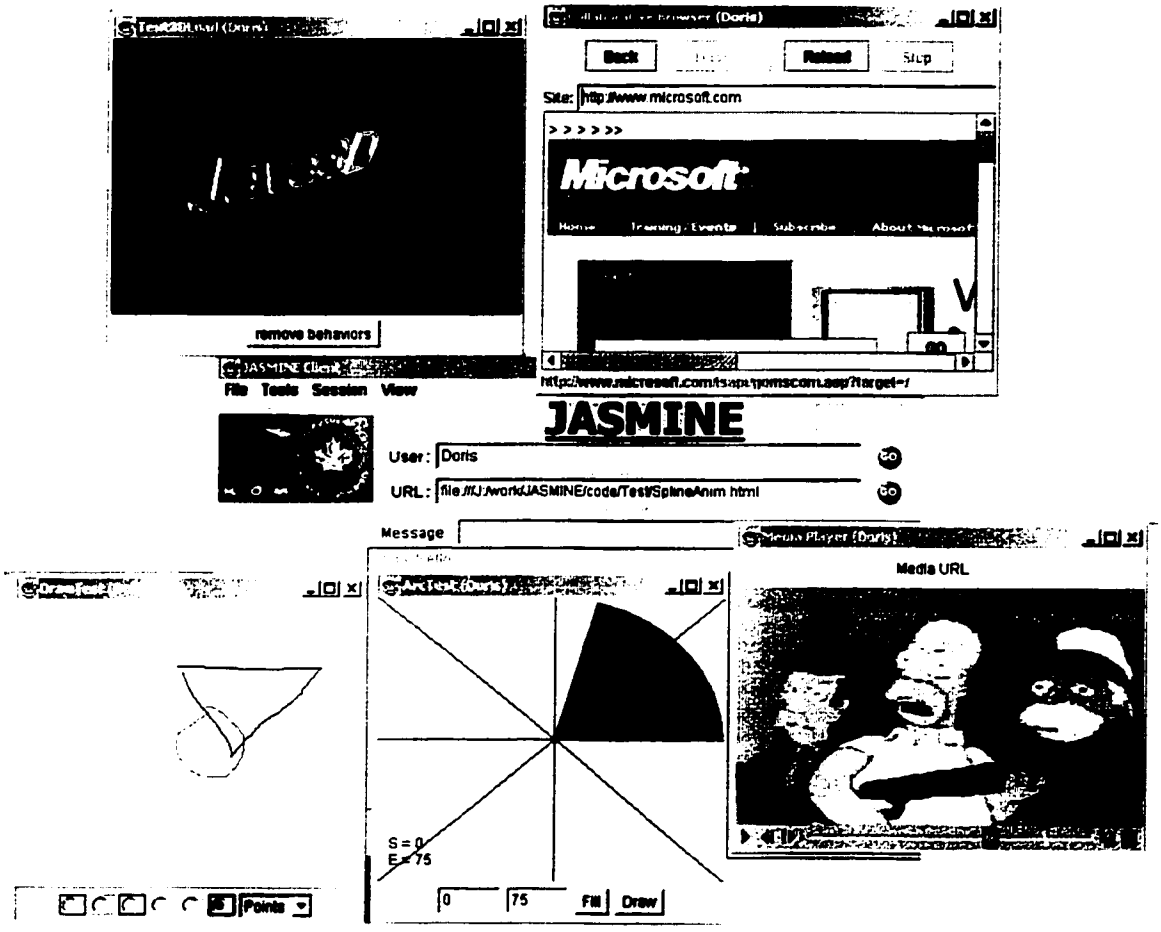

Figure 5.3 JASMINE interface with different tools

5.1.4 Collaborative VRML Viewer

Every time when users load applets to display VRML files, VRML viewers need to load Blaxxun3D class files before parsing VRML files. As a result, the speed to display files is slowed down. The system performance will slow down severely if the objects in the VRML use too much complex features, such as texture and light. The size of the 3D world is also a factor that may hurt the performance of the system.

Before using the viewer, users must clean VRML files to eliminate the content that are not supported by Blaxxun3D because Blaxxun3D supports only a part of the VRML specification. Otherwise, the browser will show error messages.

5.2 Reliability

5.2.1 Latecomer Support

In order to provide latecomer support, all the Java applications and applets to be shared have to be serializable in order for the current state to be transmitted. This is one of the constraints of the system. This constraint has limited the usage of sharing Java applications and applets. For example, Java 3D applications and applets are supported by JASMINE, but they do not support latecomers because most Java 3D classes are not serializable.

As we mentioned in the background introduction, the event-sharing paradigm has some drawbacks. One of these drawbacks is that for non-deterministic applications, it does not guarantee that all the clients see the same view at the same time. JASMINE uses the event-sharing paradigm, so it comes with this problem. For example, if the shared Java application or applet has animation functions, the latecomer will see different views from others.

For the session recorder, since events are stored in the event log without time stamps, the replay timing becomes a problem. The speed of replay depends on the speed of clients' computers.

5.2.2 Client Synchronization

The client synchronization service is optional to clients. When clients start the client software, they have a choice whether to start the echo server or not. If the system starts without the echo server, the collaboration server considers its latency zero. In this case, the system works like there is no client synchronization at all.

On the server side, because of the additional latency handler, the server is more error-prone. Sometimes problems happen when connections to some clients get terminated.

5.2.3 Multi-session Support

The multi-session support works well with the existing system. Most of the modifications are made on the server side. However, because the modifications focus on the storage format but not on the structure, the server works with no additional structure problem.

On the client side, only several new user interfaces are added for the session management, so there is no harm to the original client system functionality. The original communication module is still used to handle session management messages and it works well.

5.2.4 Collaborative VRML Viewer

Limits of Blaxxun3D become the limits of the system. All the code has to be cleaned before loaded. All the content that Blaxxun3D does not supports, such as cylinder and sphere, has to be removed. Or it has to be replaced by something that is supported by Blaxxun3D. It is very common that files created by other VRML editors, for example the Cosmo World, cannot be parsed by Blaxxun3D. Sometimes interaction with objects is not available if the VRML file is not written following standards. In this case, users can only see the objects, but they cannot interact with those objects.

5.3 Usability

In order to evaluate the usability of the system, we set a usability questionnaire to ask people who have used JASMINE to evaluate the system.

5.3.1 Questionnaire setting

The evaluation is anonymous and no participants' data are stored in any computer media. The questions we asked are in five categories, which are performance, functionality, user interface, suitability for learning and generality. Each category includes some statements about the system. The statement may be positive or negative. Participants can choose one from "Agree", "Undecided" or "Disagree".

For the statistic use, user profile information is collected. The user profile questions include participants' age range, education level, and years of experience with computers, frequency of using JASMINE and the training time they have received for using this software.

Participants are invited from both inside the university and outside the university. Some people may have used the system before and some may not. The variety of participants improves the generality of the evaluation.

5.3.2 Usability Evaluation Results

5.3.2.1 User Profile Information

There were 30 people participating in the evaluation. They were given a presentation of the JASMINE and time to practice using it. They also asked many questions regarding what they were concerned about. After a period of practice, they gave the evaluation based on what they had learned and what they thought about the system.

The age of participants ranges from 18 to over 45. 84% of them are from 25 to 34, 10% are from 35 to 45, and only one person is under 25 and one person is over 45. As to the education level, 13% of them are post-secondary graduates and 87% are graduate students. All of them have used computers for more than 3 years. Many people did not work with JASMINE in daily work and they did not have any training of using it other than the one we gave for the evaluation. However, all of them had no difficulty using the system.

5.3.2.2 Performance Evaluation Results

There are six statements in the performance category.

1. The speed to start this software is fast enough.
2. This software responds too slowly to inputs.
3. I have no difficulty in predicting how long the software will need to perform a given task.
4. The time for latecomer update is reasonable.
5. The latency handling has slowed down system performance.
6. It takes long time for the client to get history from the server.

The following chart shows the result of the evaluation.

Statement	Agree	Undecided	Disagree
The speed to start this software is fast enough	97%	3%	
This software responds too slowly to inputs	6%	13%	81%
Easy to predicting time to perform a given task	67%	30%	3%
The time for latecomer update is reasonable	87%	13%	
The latency handling slows down system	13%	33%	54%
It takes long time to get history from the server		6%	94%

Figure 5.4 Performance evaluation results

The result suggests that the performance of the system is acceptable to most users since 97% of participants answered "Agree" to the statement #1 and 81% answer "Disagree" to the statement #2. The update time for latecomers is reasonable in an intranet (the evaluation in done in an intranet) because 87% of participants think so. 54% of participants think latency handling has no influence to the system performance while 13% think it does. About 33% choose "undecided". The latency handling did slow down the system performance because we chose a small detection interval time of 10 seconds. In the real world, we are not going to use such a short interval time. Therefore, the influence on the system performance will be reduced if we carefully choose the detection interval time. The time for getting history from the server is satisfactory for most users even if we set a very short detection interval time and 97% of participants confirm this.

5.3.2.3 Functionality Evaluation Results

There are six statements in the functionality category.

1. I found the various functions in this system were well integrated.
2. The software forces me to perform tasks that are not related to my work.
3. This software occasionally behaves in a way that cannot be understood.
4. The latecomer support interrupts users' normal work.
5. The latency handling has little influence on users' work.
6. The session recorder records appropriate content.

The result of the evaluation is shown below. From the statement #1, we have learned that 84% of participants agree that all the functions are well integrated into the system. From the statement #2 and #3, we can have a general result that the system performs what it is designed to do. Statements #4, #5 and #6 are about new functions, such as latecomer support, latency handling and session recorder. Most (more than 80%) participants agree that these functions work very well with the existing system and perform required tasks.

Statement	Agree	Undecided	Disagree
Functions are well integrated	84%	6%	
The software perform tasks not related to my work	3%	33%	64%
This software occasionally behaves unexpectedly	10%	23%	67%
The latecomer support interrupts users' work	13%	6%	81%
The latency handling has little influence on work	71%	26%	3%
The session recorder records appropriate content	94%	6%	

Figure 5.5 Functionality evaluation results

5.3.2.4 User Interface Evaluation Result

There are five statements in the user interface category.

1. The way that system information is presented is clear and understandable.
2. The organization of the menus or information lists seems quite logical.
3. The software has a very attractive presentation.
4. I understand immediately what is meant by the messages displayed the software.
5. The multi-session support makes the system more complex and difficult to use.

The following table presents the statistics data of the evaluation.

Statement	Agree	Undecided	Disagree
Presentation is clear and understandable	97%	3%	
The organization of the menus is logical	97%	3%	
Very attractive presentation	81%	16%	3%
I understand immediately messages	90%	10%	
Multi-session makes the system complex	10%	16%	74%

Figure 5.6 User interface evaluation results

According to users' feedback, the user interface of JASMINE is clear and understandable. The design is also attractive. Additional menus, which are for new functions such as multi-session support and session recorder, are arranged logically.

Some users who have used old version of JASMINE felt that the multi-session makes the system complex. Compared to the old single session version of JASMINE, users now need to choose creating or joining a session before they can start working. However, this modification to the system is necessary for the system to support multiple session.

5.3.2.5 Suitability of Learning Evaluation Results

There are four statements in the suitability of learning category.

1. I needed a long time to learn how to use the software.
2. The explanations provided help me understand the software so that I become more and more skilled at using it.
3. It is easy to relearn how to use the software after a lengthy interruption.
4. In order to use the software, I must remember a great many details.

The following chart shows the result of the evaluation.

Statement	Agree	Undecided	Disagree
I need a long time to learn to use the software	20%	80%	
The explanations help to understand the software	81%	16%	3%
It is easy to relearn the software	68%	26%	6%
I must remember a great many details	3%	10%	87%

Figure 5.7 Suitability for learning evaluation results

To most people (80% of participants), it is quite easy to learn how to use the JASMINE system. The user interface is user friendly and the menu is understandable. What users need is to follow menus. They do not need to remember a lot of technical terminology and details. Even if users do not use the system for a long time, it is still easy for them to recall how to handle it. 81% of participants agree that presentations and explanations can help users to use the system.

5.3.2.6 Generality Evaluation Results

There are four statements in the generality category.

1. I found the system unnecessarily complex.
2. It is obviously that user needs have been fully taken into consideration.
3. I think that I would need the support of a technical person to be able to use this system.
4. Overall, I am satisfied with how easy it is to use this system.

For the first statement, which is "I found the system is unnecessarily complex", 84% of participants chose "Disagree". In fact, the system has clear structure and it is user friendly and easy to use. 13% of them chose "Undecided" since they were not sure. Only one person thought that the system is complex. The following figure shows people's choices.

Statement	Agree	Undecided	Disagree
I found the system unnecessarily complex	3%	13%	84%

Figure 5.8 the evaluation result for system complexity

When we designed the system, users' needs have been fully considered in JASMINE. 70% of participants agree on this. Some others, about 10% of participants, think that the system can be improved. According to the comments and feedback we got, users think it is better if the system could give prompts or explanations when they encounter problems. Moreover, they also hope that the online help could be available in the future. Nevertheless, they still agree that the system is designed under the principle that users' needs are carefully considered. The following figure gives the percentage of participants who have different choice regarding the user's needs.

Statement	Agree	Undecided	Disagree
user needs are fully taken into consideration	70%	20%	10%

Figure 5.9 The evaluation result of users needs

Technical support is necessary for any software. Most of the time, technical support is for the occasion that the system has problems. For the statement that users need technical support to user JASMINE, most people chose "Disagree". Since the system is easy to use and easy to maintain, they think it is not necessary to have technical support staff to teach them how to use it. However, other participants think that they need technical support for not only training but also for further troubleshooting or other emergency situations. When we reviewed the questionnaire design, we thought that this statement should be clearer and more specific. Nevertheless, most participants think they can deal with the system themselves.

Statement	Agree	Undecided	Disagree
Need technical support to use it	13%	10%	77%

Figure 5.10 The evaluation result of need for technical support

The final statement is to evaluate the whole system. 97% of participants are satisfied with the system and only one person was not sure if it is helpful to his work. This decision is made based their overall feeling about the system.

Statement	Agree	Undecided	Disagree
Overall, I'm satisfied with the system	97%	3%	

Figure 5.11 The evaluation result of overall satisfaction with JASMINE

5.3.2.7 Evaluation Review

Although the evaluation is given by a small group of 30 people, the result is still helpful. From the result, we can conclude that the performance of the system is acceptable, the functionality of the system is well designed, the user interface is clear and learning to use the system is easy. From the result, we learn that the system has some drawbacks and weaknesses as well. For instance, a participant hacked the JASMINE server remotely and crashed the server in a minute. This has made us to set an objective to enhance the security of the whole system.

We also received many comments and feedback from participants. These comments and feedback gave positive support to our work and practical advice that is useful for further development of the system.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

In this thesis we described the design and implementation of several important tools for transparent synchronous collaboration environments. These tools make the collaboration systems more real and generic. Compared to other similar solutions, our implementation has some significant features.

The latecomer support we developed meets the requirement of sharing Java applications, Java applets and JMF players transparently. Users can reach the current state and view the history as well. The source of the current state is provided by the moderator in a moderated session or by the starter in a non-moderated session. The current state is serialized to stream on the source side and transferred through the TCP socket connections to latecomers. Then the stream is unserialized to the original state. The tool is programmed in pure Java and it is platform independent.

Client synchronization is rarely implemented in synchronous collaboration systems. It helps to synchronize all the clients to see the same view at the same time. By testing network connections and calculating the latency of each client, the collaboration server can distribute an event according to each client's latency and ensure the event arrives at

all the clients at the same time. This tool is also written in pure Java and it is platform independent.

Many research groups are working on collaboration systems that support multiple sessions. However, their approaches mostly depend on powerful centralized servers. The session management is a heavy-duty module in their systems. In our approach, we implemented the lightweight multi-session support for a transparent synchronous collaboration system. It has complete functions to support full session management. However, it is not a separated module for the system and it can be easily migrated to other network structures, such as peer-to-peer networks.

DVE (Distributed Virtual Environments) has become an interesting research area in recent years. Some prototypes have been developed and tested. Our collaborative VRML viewer is a very simple DVE that shares VRML files transparently. Using our VRML viewer, users do not need to install any plug-in or additional software to view VRML files. The viewer is a Java applet and can be embedded into any HTML page. It can be viewed by using ordinary web browsers, such as Microsoft Internet Explorer or Netscape Navigator.

Optimizing existing synchronous collaborative systems is always the goal to people who are working in this area. With these new tools, which are latecomer support, client synchronization and multi-session support, transparent synchronous collaboration environments can provide more complete and precise support to collaboration users since users requirements are fully considered.

6.2 Future Work

Some important work still remains to be done. First, more tests on the performance need to be done with the collaborative VRML viewer. We are going to test the performance for large VRML worlds. The time for parsing a VRML world mostly depends on the number of objects, the complexity of objects and the appearance of the world. For example,

whenever users move curses in the 3D world, the viewer will repaint the world. Therefore, repainting the world becomes a bottleneck of the system.

Second, for client synchronization, we need to test it in a large network that involves more stations. These computers should locate in different geographical areas and connect to the server through different media. Here is one of the possible scenarios we might use: some users are located in different cities and connect to the server through ATM connects or phone lines; some users are located in the same city as the server and connect to the server through ISDN connections or phone lines; and some users stay in the same intranet as the server and connect to the server through 100M Ethernet cable. Since the test will involve people in different places, it will take time to finish it.

Third, we are trying to migrate the JASMINE system to a peer-to-peer network with all the services we have implemented and to compare the distributed approach with the centralized approach. The P2P approach can reduce the dependency on the central collaboration server and reduce the risk from the crash of the server as well. Theoretically, the P2P architecture does not increase the data flowing through the network, but it increases the number of network connections. Even if there are still some problems that need to be solved, we are working on them and will find solutions for them.

The application of transparent synchronous collaborative environment becomes wider with the development of the Internet and multimedia communication. To develop a qualified and practical synchronous collaboration system is always a challenge to all the researchers working in this field. The work we have done has contributed to this research and the work we will do will reach higher goals to provide more powerful, realistic and generic synchronous collaborative systems to users.

Bibliography

[1] C. W. Shiah and W. C. Chen. "A Generic Shared Window Architecture and Some Issues". Department of Computer Science and Information Engineering, National Taiwan University

[2] I. Marsic. "DISCIPLE: A Framework for Multimodal Collaboration in Heterogeneous Environments", ACM Computing Surveys, Vol. 31, No. 2es, Article No. 4, June 1999.

[3] I. Marsic and B. Dorohonceanu. "An Application Framework for Synchronous Collaboration using Java Beans". IEEE International Conference on System Sciences, January 1999.

[4] W. Wang, B. Dorohonceanu and I. Marsic. "Design of the DISCIPLE Synchronous Collaboration Framework". IASTED Internet and Multimedia Systems and Applications, October 1999.

[5] M. Ionescu and I. Marsic. "Latecomer and Crash Recovery Support in Fault Tolerant Groupware". IEEE Distributed Systems, Vol. 2, N0. 7, 2001.

[6] A. Chabert, E. Grassman, L. Jackson and S. Pietrovicz. "NCSA Habanero - Synchronous Collaborative Framework and Environment". Software Development Division at the National Center for Supercomputing Applications, 1996.

[7] L. S. Jackson. "Habanero: An Experiment in Integration Synchronous an Asynchronous Collaboration", LIS-490 ProSeminar Research Practicum, February 2000.

[8] J. Perekh. "Supporting Disconnected and Latecomer Users in the CHIME Collaborative Environment". Columbia University, 1999.

[9] T. Kindberg. "An Event-based Platform for Collaborative Object-sharing". The PerDiS Workshop on Persistence and Distribution in Java, September 1997.

[10] T. Kindberg. "Mushroom: a framework for collaboration and interaction across the Internet". CSCW & the Web, 5th ERCIM workshop, 1996, pp. 43-53.

[11] D. Fernandex, L. bellido and E. Pastor. "Session Management and Collaboration in LEVERAGE". LEVERAGE Conference, December 1998.

[12] M. Roseman and S. Greenberg. "TeamRooms: Network Places for Collaboration". CSCW96, November 1996.

[13] M. Roseman. "Managing Complexity in TeamRooms, a Tcl-based Internet Groupware Application". Fourth Annual USENIX Tcl/Tk Workshop, July 1996.

[14] H. Abdel-Wahab, O. Kim, P. Kabore and J. P. Favreau. "Java-based Multimedia Collaboration and Application Sharing Environment". CFIP '99, April 1999.

[15] H. Abdel-Wahab, P. Kabore, O. Kim and J. P. Favreau. "Replication Management of Application Sharing for Multimedia Conferencing and Collaboration". IFIP/IEEE Management of Multimedia Networks and Services, November 1998.

[16] O. Kim, P. Kabore, J. P. Favreau and H. Abdel-Wahab. "Issues in Platform-Independent Support for Multimedia Desktop Conferencing and Application Sharing". IFIP HPN'97, April 1997

[17] H. Abdel-Wahab, B. Kvande and S. Nanjangud. "Using Java for Multimedia Collaborative Applications". PROMS'96, October 1996.

[18] H. Abdel-Wahab, B. Kvande, O. Kim and J. P. Favreau. "An Internet Collaborative Environment for Sharing Java Application". IEEE FTDCS '97, October 1997

[19] J. H. Lee, A. Prakash, T. Jaeger and G. Wu. "Supporting Multi-User, Multi-Applet Workspaces in CBE". CSCL96, November 1996.

[20] Microsoft Corporation
http://www.microsoft.com

[21] F. B. Schneider. "On Concurrent Programming", Springer, New York 1997

[22] T. Axford. "Concurrent Programming: Fundamental Techniques for Real-time and Parallel Software Design". John Wiley & Sons, Chichester 1989

[23] A. El Saddik. "Interactive Multimedia Learning – Shared Reusable Visualization-based Modules". Springer, Berlin, 2001, pp. 101-132.

[24] A. El Saddik, S. Shirmohammadi, N. D. Georganas and R. Steinmetz. "JASMINE: Java Application Sharing in Multiuser Interactive Environment". IDMS2000, October 2000.

[25] S.Shirmohammadi and N.D.Georganas, JETS: Java-Enabled TeleCollaboration System", Proc.IEEE Multimedia Systems'97, Ottawa, June 1997.

[26] S.Shirmohammadi, J.C.Oliveira and N.D.Georganas, "Java-based Multimedia Collaboration: Approaches and Issues", Intern. Conf. On Telecommunications (ICT'98), Chalkidiki, Greece, June 1998

[27] Blaxxun Corporation
http://www.blaxxun.com

[28] VRML Specification

http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[29] G. Chung, P. Dewan, and S. Rajaram. "Generic and Composable Latecomer Accommodation Service for Centralized Shared Systems". IFIP Working Conference on Engineering for Human-Computer Interaction, September 1998.

[30] G. Chung, K. Jeffay and H. Abdel-Wahab. "Accommodating Latecomers in Shared Window Systems". IEEE Computer January 1993, Vol. 26, No. 1, pp. 72-74

[31] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing". IEEE/ACM Transactions on Networking, December 1997, Volume 5, Number 6, pp. 784-803.

[32] J. Vogel, M. Mauve, W. Geyer, V. Hilt and C. Kuhmunch. "A Generic Late-join Service for Distributed Interactive Media". ACM MM 2000, pp. 259-268, October 2000.

[33] T. Illmann, R. Thol and M. Weber. "Transparent Latecomer Support for Web-Based Collaborative Learning Environments". CSCL2002, January 2002.

[34] P. Bhandarkar. "Replicating Distributed Events for Real-Time Collaboration". Special Problems Research report. Department of Electrical and Computer Engineering. Rutgers University, 1998

[35] P. T. Conrad and B. Greenstein. "Teaching Network Performance Measurement Using Java When The Students Don't Already know Java". Department of Computer and Information Sciences, Temple University

[36] B. Dorohonceanu and I. Marsic. "A Desktop Design for Synchronous Collaboration". Graphics Interface `99, June 1999, pages 27-35.

[37] B. Dorohonceanu, B. Sletterink and I. Marsic. "A Novel User Interface for Group Collaboration". IEEE System Sciences, January 2000.

[38] B. Driggers, J. Alameda and K, Bishop. "Distributed Collaboration for Engineering and Scientific Applications Implemented in Habanero, a Java-Based Environment". http://union.ncsa.uiuc.edu/habenaro.

[39] F. Frank, N. Baloian, U. Hoppe and E. Reimberg. "MatchMaker: Synchronizing Objects in Replicated Software-Architectures".IEEE CRIWG 2000.

[40] D. Herlea. "Computer Supported Collaborative Requirements Negotiation". KAW98, April 1998.

[41] M. Hohmuth and H. Hartig. "Pragmatic Non-blocking Synchronization for Real-time Systems". USENIX Annual Technical Conference, June 2001.

[42] P. L. Isenhour, J. B. Begole, W. S. Heagy and C. A. Shaffer. "Sieve: A Java-based Collaborative Visualization Environment". IEEE Visualization '97, October 1997.

[43] J. Jogel, M. Mauve, W. Geyer, V. Hilt and C. Kuhmunch. "A Generic Late-Join Service for Distributed Interactive Media" ACM Multimedia 2000, October 2000.

[44] T. Kindberg, G. Coulouris, J. Dollimore and J. Jeikkinen. "Sharing Objects over the Internet: the Mushroom Approach". IEEE Global Internet, November 1996.

[45] D. Li. "Sharing Single-user Editors by Intelligent Collaboration Transparency". ACM GROUP'01 workshop on collaborative editing systems (IWCES-3), Sept.30, 2001.

[46] S. Lukosch and C. Unger. "Flexible Synchronization of Shared Groupware Objects". ACM GROUP'99 Workshop on Consistency Maintenance and Group Undo in Real-Time Group Editors, November 1999.

[47] M. A. Mora and R. Moriyon. "Collaborative History Management for Chess and Tutoring Systems". First Technical Workshop of the Computer Engineering Department, March 2000.

[48] H. Ogata, K, Matsuura and Y. Yano. "Synchronizing Group Interactions with Lecturing Video in Agent-based Asynchronous Virtual Classroom". CSCL 2000, January 2002.

[49] C. Potts, J. D. Bolter and A. Badre. "Collaborative Pre-writing with a Video-based Group Working Memory". GVU Technical Report No. GIT-GVU-93-35, 1993.

[50] T. Rist, J. C. Martin, F. D. Neel and J. Vapillon. "On the Design of Intelligent Memory Functions for Virtual Meeting Places: Examining Potential Benefits and Requirements". Journal Le Travail Humain, Vol. 63 No. 3/2000, pp. 203-225.

[51] J. Roth and C. Unger. "Developing Synchronous Collaborative Applications with TeamComponent". COOP2000, May 2000, IOS Press, pp.353-368.

[52] V. Roussev, A. Sharama and P. Dewan. " COMP 290: Collaboration Systems. A Collaborative Spreadsheet". Department of Computer Science, The University of North Carolina at Chapel Hill, April 1997.

[53] D. C. Schmidt and T. Suda. "Experiences with an Object-Oriented Architecture for Developing Dynamically Extensible Distributed System Management Software". The Conference on Global Communications (GLOBECOM), pp. 500– 506, IEEE, November 1994.

[54]S.Shirmohammadi and N.D.Georganas. "Collaborating in 3D Virtual Environments: A Synchronous Architecture". IEEE 9th International Workshops (WETICE) on Knowledge Media Networking workshop, June 2000.

[55] D. D. Suthers. "Architectures for Computer Supported Collaborative Learning". The IEEE International Conference on Advanced Learning Technologies (ICALT), August 2001.

[56] D. B. Terry, K. Petersen, M. J. Spreitzer and M. M. Theimer. "The Case for Non-transparent Replication: Examples from Bayou". IEEE Data Engineering Bulletin, Volume 21, Number 4, pp. 12-20, December 1998.

[57] P. Torlind, M. Stenius and M. Johanson. "Collaboration Environments for Distributed Engineering Development of a Prototype System". CSCWD `99, September 1999.

[58] M. B. Twidale, D. M. Nichols and C. D. Paice. "Browsing is a collaborative Process". Information Process & Management, 1997, 33(6), 761-83.

[59] J. Vogel, M. Mauve, W. Geyer, V. Hilt and C. Kuhmunch. "A Generic Late-join Service for Distributed Interactive Media". ACM MM 2000, pp. 259-268, October 2000.

[60] J. Zumbach, M. Muehlenbrock, M. Jansen, P. Reimann and H. U. Hoppe. "Multi-dimensional Tracking in Virtual Learning Teams: An Exploratory Study". The Conference on Computer Supported Collaborative Learning CSCL 2002 pp. 650-651, January 2002.

[61] Java Specification
http://java.sun.com