In International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services 2000 (IDMS 2000), Enschede, Netherlands, pages 214-226. Springer, October 2000

JASMINE: Java Application Sharing in Multiuser INteractive Environments

Abdulmotaleb El Saddik[1], Shervin Shirmohammadi[2], Nicolas D. Georganas[2], and Ralf Steinmetz[1,3]

1) Industrial Process and System Communications, Dept. of Electrical Eng. & Information Technology, Darmstadt University of Technology, Darmstadt, Germany
2) Multimedia Communications Research Laboratory, School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada
3) GMD IPSI, German National Research Center for Information Technology, Darmstadt, Germany
{Abdulmotaleb.El-Saddik, Ralf.Steinmetz}@kom.tu-darmstadt.de
{Shervin.Shirmohammadi, Nicolas.Georganas}@mcrlab.uottawa.ca

Abstract. In this paper, we describe an approach for *transparent* collaboration with java applets. The main idea behind our system is that user events occurring through the interactions with the application can be caught, distributed, and reconstructed, hence allowing Java applications to be shared transparently. Our approach differs from other collaborative systems in the fact that we make use of already existing applets and applications in a collaborative way, with no modifications to their source-code. We also prove the feasibility of our architecture presented in this paper with the implementation of the JASMINE prototype.

1 Introduction

The simplicity of access to a variety of information stored on remote locations led to the fact that the World Wide Web has gained popularity over the last decade. In this context, Computer Supported Collaborative Learning (CSCL) is becoming more and more important. Collaborative systems allow users to view and interact with a distributed application during a session. The use of collaborative systems increases in research and business as well as in education. A problem of many cooperative applications is their platform dependence, leading to the fact that users communicating in heterogeneous environments are restricted in their choice of a cooperative application. For example, a user might choose a UNIX-workstation, while another might prefer Windows 95/98/NT or a Macintosh. The introduction of the platform-independent programming language, Java, made it possible to overcome these problems. Diverse approaches were used to develop Java-based collaborative systems. Almost every system described in the literature requires the use of an Application Programming Interface (API) [3], [4], [5]. Others are trying to replace some Java-components with self-defined collaborative components in a transparent manner [7].

The approach presented in this paper differs from other approaches in the way that we neither propose a new API for developing collaborative systems nor try to replace core components at run time. In fact, a great variety of well-designed applets already exist on the World Wide Web, which were developed to be run as stand-alone and it would not be acceptable or possible for many developers to re-implement or change these programs to make them work in a collaborative way. In our architecture, we make use of the Java Events Delegation Model [13] to extend the capabilities of Java applications in a way that stand-alone applets can be used collaboratively. The delegation event model of JDK1.1 provides a standard mechanism for a source component to generate an event and send it to a set of listeners. Furthermore, the event model also allows to send the event to an adapter, which then works as an event listener for the source and as a source for the listener. Because the handling of events is a crucial task in developing an application, this enhancement makes the development of applets much more flexible and the control of the events much more easy.

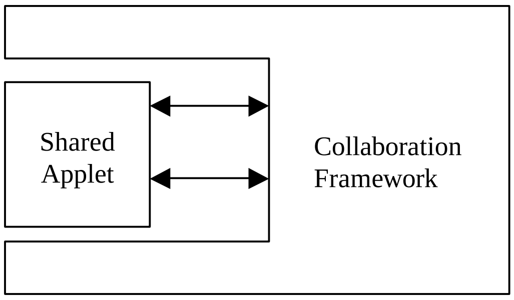


Fig. 1. Illustration of the main idea

The approach behinds our concept, which is illustrated in Figure 1, underlies the following requirements:

No restrictions in the source code are required to share an applet. Both AWT- and Swing-based applets should be supported. A solution restricted to only one kind of components is not acceptable.

Applications using the standard Java-Core API should be supported.

No new API should be developed.

As less as possible of the network's bandwidth should be consumed.

The practicality of our architecture is proven by an implementation. We have developed a collaboration system, called JASMINE (Java Application Sharing in Multi-user INteractive Environments), which facilitates the creation of multimedia collaboration sessions and enables users to share Java applets and applications, which are either pre-loaded or brought into the session “live”. The system also provides basic utilities for session moderation and floor control. Our approach applies to both applets and applications and hence these terms are sometimes used interchangeably in this document.

The rest of the paper is organized as follows. Section 2 discusses the system architecture, while section 3 describes the implementation of JASMINE, followed by discussion of related work in section 4. Finally, section 5 concludes the paper and gives an out-look for future work.

2. JASMINE Architecture

The principal idea of JASMINE is that user events occurring through the interaction with the GUI of an applet can be caught, distributed, and reconstructed, hence allowing for Java applets to be shared transparently. This form of collaboration which is supported as long as a learning-session takes part, enables users to interact in real-time, working remotely as a team without caring about low-level issues, such as networking details.

Figure 2 illustrates the overall concept of JASMINE, where our collaboration framework wraps around an applet that is to be shared. The framework listens to all events occurring in the graphical user interface of the applet and transmits these events to all other participants in order to be reconstructed there. The framework captures both AWT-based and Swing-based events. After capturing the event, it is sent to the communication module (JASMINE-Server) where the event is sent to all other participants in the session.

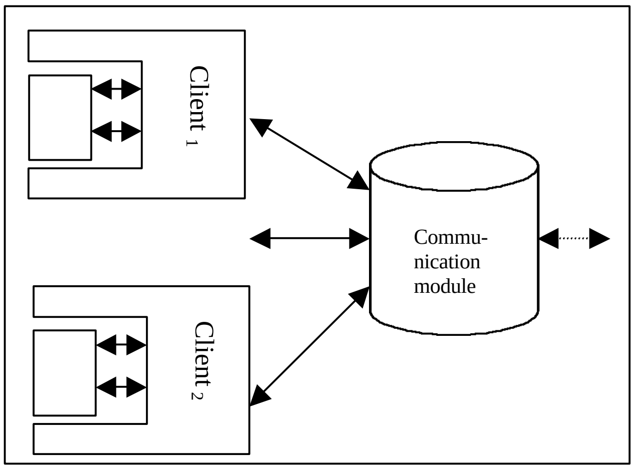


Fig. 2. Overall system architecture of JASMINE

In the next sections we are going to discuss the architecture in more details, first the client side, and then the communication module.

2.1 JASMINE-Client

The JASMINE client can be seen as a component adapter. Every event occurring at the graphical user interface of the application is sent to this adapter, which then sends the events to the collaboration server (JASMINE-Server). The client is a Java application, which consists of the following components:

Collaboration Manager
Component Adapter
Listener Adapter
Event Adapter

These components are discussed next.

2.1.1 Collaboration Manager

The Collaboration Manager is the main component on the client side and provides the user with a graphical interface offering options such as joining the session, starting and sharing applications/applets and chatting with other participants. The collaboration manager is also responsible for dispatching external events coming from the communication module and forwarding them to the component adapter, as well as receiving internal events from the component adapter and sending them to the communication module.

2.1.2 Component Adapter

The Component Adapter maintains a list of the GUI-components of all applications and applets. This list is created with the help of the *java.awt.Container* class, which allows us to get references of all applet components [13]. With the help of the main window of an application, a list of the GUI components in the application can directly be created. Therefore, the main window of an application loaded by the Collaboration Manager is registered by the Component Adapter. However, Java applets do not use stand-alone windows. They are an extension of the class *java.applet.Applet* and thus of *java.awt.Panel*. Hence, applets can be easily placed into a window, which can then be registered as the main window for the applet. All these registrations are done at the Component Adapter. An example syntax of the registration by the Component Adapter is shown in Figure 3.

```
.....
Class cl = Class.forName(className);
// If it is an applet, instantiate and locate
// it in a Frame
myApplet = (Applet)cl.newInstance();
myApplet.init();
myWindow = new Frame("Titel");
myWindow.add("Center", myApplet);
// Otherwise (if it is an instance of Window) just
// instantiate it
myWindow = (Window)cl.newInstance();
// Register this Frame as main Frame
// by Components Adapter
ComponentsAdapter.addContainer(myWindow);
....
```

Fig. 3. Excerpt of the instantiation method

After the registration is completed, a list of all Swing and/or AWT-components within the loaded application/applet is created. This task is done in the same order on each client, so that a component has the same reference identification at all clients. These references are used to point to specific components, which are the source of the events generated internally and the recipient of the events generated externally. With the help of the references, the recipient of an incoming event is located and the event is reconstructed on each client, as if it occurred locally.

2.1.3 Listener Adapter

The Listener Adapter implements several AWT listeners, which listen to *MouseEvent* and *KeyEvent* for all AWT-components except of *java.awt.Scrollbar*, *java.awt.Choice* and *java.awt.List*. For these components the Listener Adapter listens to *AdjustmentEvent*, *ItemEvent* and *ActionEvent*. When an event occurs on the GUI of the application, the Listener Adapter catches it, converts it to an external event, and forwards it to the Collaboration Manager. The Collaboration Manager in turn sends this event to the communication module, which propagates the event to all other participants.

2.1.4 Event Adapter

The Event Adapter works opposite to the Listener Adapter: it converts incoming external events to AWT events, which then can be processed locally.

2.1.5 Data Flow

Let us summarize the client side's architecture through the following data flow diagram. Figure 4 shows the overall event circulation of the system.

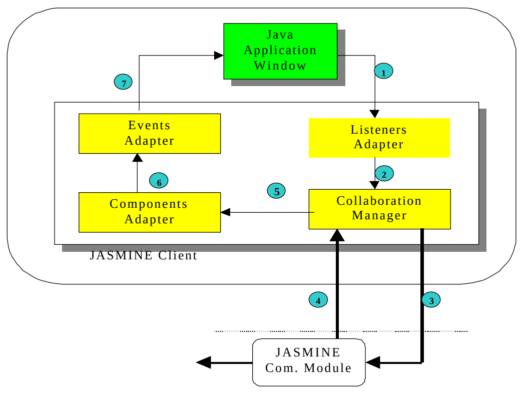


Fig.4. Event Circulation

There are two main data paths in the system: the first path is labeled with numbers 1,2 and 3. This path is used to send the internal AWT events to the communication module, and it works as follows: any Event occurred in a Java-application is caught by the Listener Adapter. The Listener Adapter first tests whether the event is an external or an internal event. It then sends only the internal events, which were not received from other clients, to the Collaboration Manager, which in turn sends the events to the communication module.

Via the second data path shown in figure 4 with numbers 4, 5, 6 and 7, the external AWT events received from the communication module are captured by the Collaboration Manager and the Component Adapter in order to reconstruct the event locally. After receiving the remote event, the Component Adapter extracts the information about its target component and sends this information together with the events to the Event Adapter. The Event Adapter converts the event to normal AWT events and sends them to the application, which then reacts to the event in the same manner as it would to a local user's interaction with the application's GUI.

2.2 JASMINE Server

JASMINE uses a multithreaded server, where the main server launches a sub-server for each user joining the session. The sub-server is responsible for processing only the update messages or requests coming in from its own client. Once the sub-server receives the update message, it will send it to all other clients in the session (figure 5). This will create a fast system response, at the expense of more resources utilized due to sub-server threads. However, usually only one client at a time can control and interact with an application (due to floor control as we will see), and most threads will simply be waiting and won't consume too many resources.

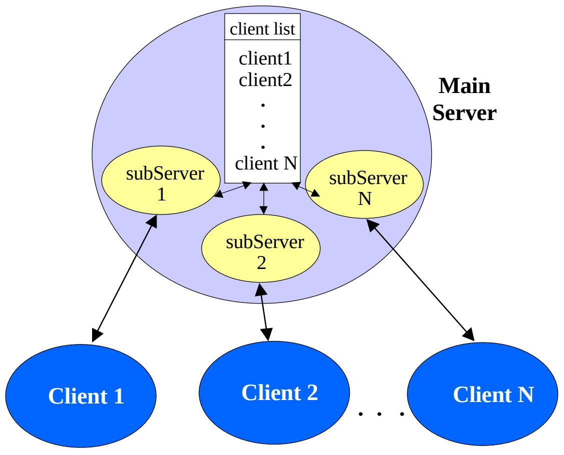


Fig. 5. JASMINE Server

The server's main job is to propagate the incoming events from a user to all other users. But it also provides other services, which are necessary for maintaining a collaboration session. It provides services for session moderation and management, floor control, and data exchange. Data exchange is of particular importance for multimedia sessions as we will see next.

2.3 Advanced Multimedia Applications

As discussed in the literature, a pure transparent collaboration system is not sufficient for multimedia applications [12]. This fact is due to specific services that are required by multimedia applications such as synchronization, quality of service, etc. For example, think of a collaboration session where a video applet is being played. When one user presses the pause button, simply capturing the "pause event' and sending it to all other clients is not sufficient because when other clients receive the pause event and apply it to their video player, at each client the video player will pause on a different frame and clients will not be synchronized. Hence there is a need to send control messages between clients, such as "pause on frame number 57" to maintain consistency among all users. The JASMINE server provides a high-level API that can be used for this type of advanced requirements. However, an application must specifically use the API to take advantage of these functionalities, hence the transparent feature of the system is somewhat diminished.

3 Implementation

Figure 6 illustrates a sample screenshot of a typical JASMINE session. It shows the client's Collaboration Manager and some shared applets and applications running in the session.

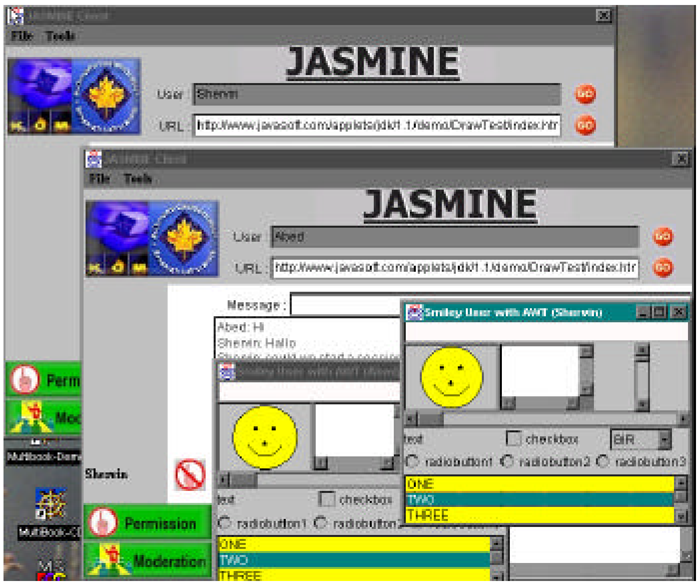

User : Sherwin
URL : http://www.javasoft.com/applets/jdk/1.1/demo/DrawTest/index.htr
User : Abed
URL : http://www.javasoft.com/applets/jdk/1.1/demo/DrawTest/index.htr
Message :
Abed: Hi
Sherwin: Hallo

Fig. 6. A screenshot of a sample JASMINE session

3.1 Configuration file

Information about locally available applications and applets, which can be used in a collaborative way, are read from a configuration file. The configuration file, which is organized as a properties file, contains the names of the applications/applets, which will be presented in the menu and the full names of their main class or URL. The entries have the following syntax:

```
    application.[n].name = [name]
    application.[n].class = [class]
```

where:

n: number of the application in the list.
name: a suitable name for the application to be shown in the menu.
class: full name of the main class.

An example configuration file is illustrated in Figure 7.

```
#Application entries
application.1.name=myTestApplication
application.1.class=kom.develop.apps.MyApp
# Applet entries
applet.1.name=myTestApplet
applet.1.class=kom.develop.applets.TestApplet
# URL entries
url.1.name= TestUrl
url.1.address=http://desiered.server/test.html
```

Fig. 7. Excerpt from a configuration file

Before starting the session, applets and applications that are thought to be useful can be placed in this configuration file. Additional applets and applications can be brought into the session live as needed by typing the corresponding URL in the appropraite field.

3.2 Floor Control

A collaborative system must address many issues such as synchronization, latecomers, management or moderation, floor control, and awareness [12]. Among these, floor control is perhaps the most primary issue without which a collaborative session won't function properly. In short, floor control ensures that only one person at a time controls the shared application. Without floor control, there will be collisions of events, which leads to unwanted results in the shared application.

In JASMINE, floor control is achieved by means of locking. Each application has a corresponding *semaphore* on the server. When a user wants to interact with the shared application, the system first locks the application by locking a semaphore. At this point, any other users trying to interact with the application will be denied access. When the first user is finished, the system releases the semaphore and others can take control of the application.

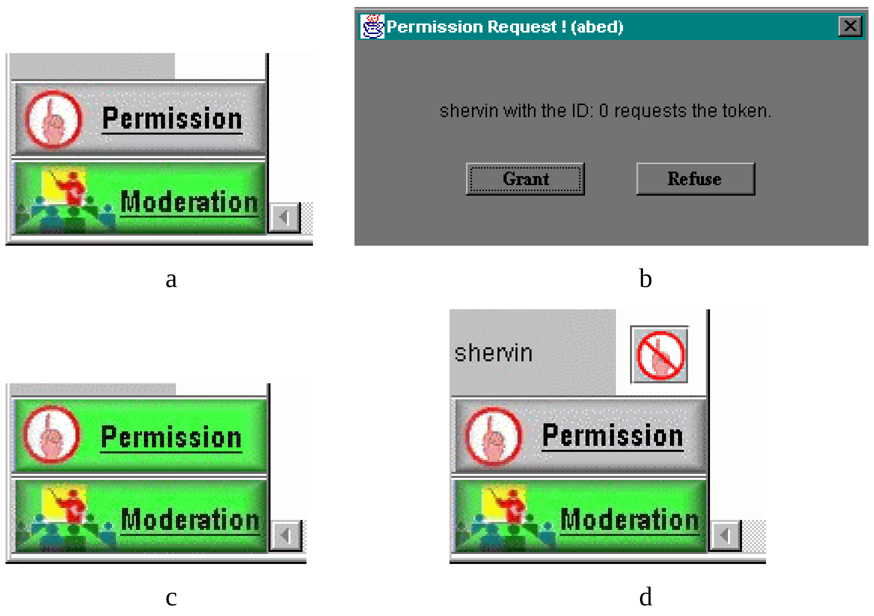
```
public void mouseDragged(MouseEvent e) {
  //user is dragging the mouse, so ask for control
  if (getControl()==true) {
    // do whatever must be done for a mouse drag
    releaseControl();
  }
  else displayMessage("Access Denied!");
}
```

Fig. 8. Intuitive floor control

For a specific shared application, most developers prefer an “intuitive” implementation of the floor control capability; i.e., as soon as the user tries to interact with the application, the client automatically asks for floor control and allows or disallows its user to interact. After the user is finished, the client releases the lock automatically. Figure 8 shows sample Java code that demonstrates how the floor control is used in an intuitive way. This approach is in contrast to the “direct” approach, where a client must specifically ask for control, for example by pressing a "control-request" button.

3.3 Moderation

Although floor control addresses the issue of event collisions, it works on a first-come-first-serve basis. This in turn leads to the possibility of a participant to abuse or disrupt the session by feeding unwanted events into the session. There is therefore a need to have a moderator in order for a session to be more productive, for example, a teacher moderating a distance learning session. The moderator is usually the person who calls for a collaborative session and starts the server. In JASMINE we have two types of sessions: moderated, and non-moderated. The server can be started by specifying a login name and password for the moderator.



shervin with the ID: 0 requests the token.

Fig. 9. Moderation capabilities in JASMINE.

Once the session starts, the moderator can login at any time and take control of the session. When the session is moderated, no one can send any events to the server. A participant wishing to do so must ask for permission from the moderator as shown in Figure 9a. The moderator will subsequently receive a message indicating the participant's request to interact (Figure 9b) which the moderator can allow or refuse. Upon moderator's acceptance of the user's request, the user will receive a green light, which indicates that he or she can now send events to the session (Figure 9c). The moderator can also dynamically "cut off" a user's permission to interact if needed (Figure 9d). In JASMINE, we allow only one user at a time to have permission to send events, although this number can be increased based on the application.

4 RELATED WORK

There are many Java-based collaboration systems, none of which offer a management or moderation feature similar to ours. Kuhmünch [10] at the University of Mannheim developed a Java Remote Control Tool, which allows the control and synchronization of distributed Java applications and applets. The drawback of this approach is that it is necessary to have access to the original source code of the application or applets in order to make it collaborative. That means every applet must initiate a Remote-Control-Client object, which is usually done in the constructor of the applet. Also, the event handling within the applet must be modified in order to receive and/ or send events from / to remote applets. The Java Shared Data Toolkit (JSDT) from JavaSoft is also an API-based framework [15]. Habanero [1] is an approach that supports the development of collaborative environments. Habanero is in its terms a framework that helps developers to create shared applications, either by developing a new one from scratch or by altering an existing single-user application which has to be modified to integrate the new collaborative functionality. Instead of using applets, which can be embedded in almost every browser, the Habanero system uses so-called “Happlets” which need a proprietary browser to be downloaded and installed on the client site. Java Collaborative Environment (JCE) has been developed at the National Institute of Standards and Technology (NIST) coming up with an extended version of the Java-AWT [2] called Collaborative AWT (C-AWT). In this approach AWT-components must be replaced by the corresponding C-AWT components [3].

All these approaches propose the use of an API, which has the cost of modifying the source-code of an application, re-implementing it or to design and implement a new application from scratch in order to make it collaborative.

Java Applets Made Multiuser (JAMM) [8] is a system, which is similar to our approach in terms of its objective: the transparent collaboration of single-user applications. The difference between JAMM and JASMINE is the way collaboration is achieved. In JAMM [6], the set of applications that can be shared is constrained to those that are developed using Swing user interface components as part of Java Foundation Classes, which are part of the standard JDK since version 1.2. JAMM’s set of applications is furthermore restricted to those which implement the Java serializable interface.

5 Conclusion

We presented the architecture and implementation of our transparent collaboration framework for Java applets and applications. We developed this architecture in order for users to be able to collaborate via collaborative-unaware applications and applets without modifying the source code. Our architecture enables us to use almost all single-user applets and applications in a collaborative way. We have successfully tested our system on a number of applets. We also observed that using the TCP-client-server approach of our communication module can support relatively large number of users.

There are two outstanding issues remaining. These issues are not directly related to JASMINE but are research areas of the transparent collaboration paradigm. The first issue is that of latecomer-support. When a user starts a session later than other participants, there is a need to bring this user up-to-date as opposed to start from scratch. This can be achieved either by sending the entire object state of the shared application to the newcomer using object serialization, or by sending all the events occurred up to now to the new user so that it follows the same sequence of events that other participants have gone through [12]. We're currently using JASMINE to experiment with these methods.

Another issue was brought up in Section 2.3: multimedia inter-client synchronization and control. Transparent collaboration cannot address this issue alone and we believe that using an API is necessary to achieve such functionality for multimedia applications.

Today, computing environments where Java applications and applets are running over IP have become very popular and widespread. Our architecture helps people to collaborate in such environments easier.

ACKNOWLEDGMENTS

The authors acknowledge the financial assistance of the Volkswagen Stifftung, Germany, as well as the Telelearning Network of Centers of Excellence Canada (TL-NCE) and the Natural Sciences and Engineering Research Council Canada (NSERC).

REFERENCES

1. Chabert et al, , "Java Object Sharing in Habanero", Communications of the ACM, Volume 41, No. 6, June 1998, pp. 69-76.
2. H. Abdel-Wahab et al “An Internet Collaborative environment for Sharing Java Applications” IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems (FTDCS'97), October 29 - 31, 1997, pp. 112-117.

3. H. Abdel-Wahab et al, *"Using Java for Multimedia Collaborative Applications"* Proc. PROMS'96, Madrid, Spain, 1996.
4. *Handheld IP Connectivity for 1998*, IEEE Internet Computing, Vol. 2, No. 1, January/February 1998, pp. 12-14.
5. International Data Corporation, "IDC's Forecast of the Worldwide Information Appliance Marketplace 1996-2001", IDC Bulletin #w15080, December 1997, (screen phone revisions 5/7/98).
6. Abdulmotaleb El Saddik, Oguzhan Karaduman, Stephan Fischer, and Ralf Steinmetz. "Collaborative Working with Stand-Alone Applets". In Proc. of the 12th International Symposium on Intelligent Multimedia and Distance Education (ISIMADE'99), August 1999.
7. J. Begole et al, "Leveraging Java Applets: Toward Collaboration Transparency in Java", IEEE Internet Computing, March-April 1997, pp. 57-64.
8. J. Begole et al, "Transparent Sharing of Java Applets: A Replicated Approach". Proc. Symposium on User Interface Software and Technology, ACM Press, NY, 1997, pp. 55-64.
9. J. Grudin, "Computer-Supported Cooperative Work: History and Focus", IEEE Computer, Vol. 27, No. 5, May 1994, pp. 19-26.
10. Kuhmünch et al, "Java Teachware - The Java Remote Control Tool and its Applications", Proc. ED-MEDIA'98, 1998.
11. Multimedia Communication Forum Inc., "Multimedia Communication Quality of Service", MMCF document MMCF/95-010, Approved Rev 1.0, September 24, 1995.
12. S. Shirmohammadi et al, "Applet-Based Telecollaboration: A Network-Centric Approach", IEEE Multimedia, Vol. 5, No. 2, April-June 1998, pp. 64-73.
13. Stephan Fischer and Abdulmotaleb El Saddik, Open Java: Von den Grundlagen zu den Anwendungen. Springer-Verlag, ISBN: 3540654461 (1999).
14. K. Obraczka, "Multicast Transport Protcols: A Survey and Taxonomy", IEEE Communications, Vol. 36, No. 1, 1998, pp. 94-102.
15. Javasoft (for Java, JINI, RMI, and JSDT technologies) http://www.javasoft.com