Java-Based Multimedia Collaboration: Approaches and Issues

Shervin Shirmohammadi, Jauvane C. de Oliveira*, and Nicolas D. Georganas
Multimedia Communications Research Laboratory
School of Information Technology and Engineering
University of Ottawa, Canada
[shervin | jauvane | georgana]@mcrlab.uottawa.ca

ABSTRACT

Real-time collaboration systems, in which participants share multimedia documents and applications in real time, have been a subject of interest for many years. Although computer supported cooperative work (CSCW) systems have existed for a long time, Web-based collaboration tools, such as Microsoft NetMeeting, have started to emerge relatively recently. Moreover, there has been much advancement in Internet-related technologies lately. Among them, Java applets seem to have introduced the most exciting notion: platform-independent applications provided by the network. From multimedia collaboration point of view, the next step in the evolution of applets is their real-time sharing among many users. In this paper, we will discuss the design issues of a framework for multimedia collaboration using Java applets. We will also elaborate on the implementation matters encountered during our experimentation with our JETS prototype.

I. INTRODUCTION

The introduction of Java as a platform-independent programming language for the Internet has had a significant impact on the related technologies. By embedding Java applets in HTML documents, the applications are now going to the user in the form of applets, instead of the traditional case of the user finding and installing the applications. Users have only to make sure they have a Java-enabled Web browser on their platform. Today, Java is considered by many experts as the de-facto programming language for Internet computing. Java's Web-accessibility and platform-independence are its most important properties which set it apart from other type of programs.

Almost parallel to the Java paradigm, the subject of Web-based telecollaboration has also recently received a lot of attention. Although computer supported cooperative work (CSCW) systems have existed for a long time [6], Web-based collaboration tools that permit sharing of multimedia applications among participants on the Internet have recently started to emerge. Microsoft NetMeeting, Vocaltec ICP, and Netscape Collaborator are only a few of these types of tools. There are also some Java-based collaboration tools available such as the NCSA Habanero and the Java Collaborative Environment (JCE) which allow sharing of basic multimedia applets such as whiteboards; however, very few of these tools use applets as their base for collaborative applications. Although applets can usually be converted to applications and be shared using conventional techniques, there is a fundamental difference between sharing applets and applications:

In practice, applets can be shared through a Web browser or a network station. This is not only platform-independent but also requires no downloading and installation of the specific application on the user side. A user simply navigates to a given URL and joins a session. Furthermore, users will always have the latest version of the applet. This is not the case with applications. Generally, applications cannot be shared using a Web-browser. This means that a package must be downloaded and installed on the machine of every user who wants to take part in the collaboration session; an approach which is very similar to collaboration using non-Java applications. Hence, the emphasis in the application sharing method is the portability feature of Java, whereas the applet sharing method is more a network-centric approach.

In this article, we present a client-server architecture for sharing Java applets. We will show the collaboration issues that we came across based on our experience with our Java-Enabled Telecollaboration System (JETS) prototype and its applets.

II. COLLABORATION ARCHITECTURE

There are two main architectural approaches for telecollaboration: centralized and distributed architecture. The trade-off is between simpler clients, avoiding complex distributed algorithms, and ease of implementation (centralized); as opposed to faster processing and avoiding an extra bottlenecked entity performing the duties of a server (distributed). We believe that in the case of Java applets, using a central server is more easily justifiable than using a non-centralized architecture.

In order to join the session and start the applets, the client machines have to download the Java applets from a central server. This means that a server already exists and is not an additional resource utilized by the system; it is very necessary and part of the system to start with. Furthermore, as part of Java's security restrictions, applets can only make network connections back to the server from which they were downloaded. Although future releases of Java will have more flexibility by means of allowing the user to give more permission to a

* Sponsored by CAPES, the Brazilian Ministry of Education Agency: Bolsista da CAPES - Brasilia/Brasil

Java applet, it is still recommended to write applets that can work in any environment. Hence, for Java applets, the use of a server becomes almost mandatory in order for the client-to-client communication to take place. In addition, as we will see next, there are many issues that are solved more easily with a server-based approach than one without a central server.

III. IMPLEMENTATION ISSUES

Figure 1 shows a simple client-server system with identical client applets. When one user interacts with the applet, by for example clicking a button, one or more events are generated. Each event is sent to the server, which multicasts it to all other clients. The other clients then process the event as if it was generated locally and therefore recreate the intended actions of the original client.

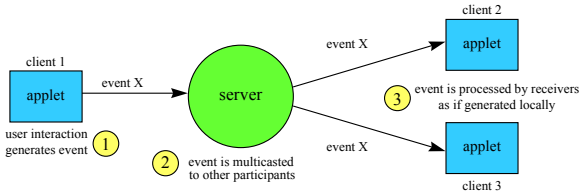

Figure 1. A simple client-server architecture for event-multicasting.

One can think of many issues that a multi-user collaboration environment such as that in figure 1 must address. The main issues are latecomers, participants' awareness, management, event collision, and synchronization.

A. Latecomers

When a user joins a session already in progress, the user must be presented with the current states of the shared applets. Using a central server, the newcomer can receive these states in two main ways. One way is to play back the sequence of events occurred on an application. This is referred to as event logging [10]. This not only requires huge buffers, but also might be unstable due to an expired entity such as a temporary file which was erased. The other approach is to keep track of the current object state of the applications and send them to the newcomer. In case of Java, this can be easily achieved by means of Java's *object serialization*. Java Object Serialization can transparently send objects with their current state over the network. However, this requires that the server not only multicast user events, but also keep a copy of the application object and execute the received events on it in order to maintain the application's current state up to date.

B. Awareness

Awareness is the ability of a given participant of a collaborative session to have feeling of both existence and actions of the others. According to the literature [1] there are as many as eleven different elements that provide such "feeling" including *presence*, *activity level*, *actions*, *intentions*, *changes*, and *abilities*.

This issue has been of interest for many years in the field of teleconferencing systems and virtual reality. In teleconference systems, video is used to watch the other participants and to be aware of their existence and actions. The awareness in VR and 3D simulation systems is usually implemented through an avatar that tries to mimic the behaviors and actions of its user; other participants can then see the user as in the real world. In non-3D environments; however, this technique is inadequate.

To achieve awareness, a system can use either explicit mechanisms, such as direct communication, or indirect ones, such as simple observation of others work by noticing the effect of their actions. This can be improved by the use of a telepointer as used in, for example, Microsoft NetMeeting. When a user moves the mouse over an application, the mouse movement is also reflected on the screen of other participants. However, if there are many users, this method needs some enhancements. Let's say there are 50 users, and all of them are allowed to interact with the application at the same time; naturally there will be too many pointers on the screen to keep track of. A simple solution for this, as used in NetMeeting, is to allow only one user at a time to have access to the application which is usually achieved by some sort of locking mechanism [3].

Again, the use of a central server proves to be beneficial for this feature. The server can keep track of all participants and relay them to a newcomer, or notify others when a user leaves the session.

C. Management

Often overlooked by existing implementations, a session manager is a key component of a functional telecollaboration system. There is a need for a chairperson who controls participants' access rights for different applications. This person can dynamically set individual user's access rights such as *no-access*, *view-only*, and *view/interact*. For instance, whenever a user abuses his/her access rights by taking control over the application longer than allowed, the chairperson can intervene by changing the user's access rights. Although not many, some collaboration/conferencing systems today implement session management. The TVS system, for example, implements a management system that controls participants' access rights both manually and automatically [7].

A central server can facilitate the implementation of this issue. Users' access rights can be predetermined by the session manager. These rights can be categorized into generic rights such as *guest*, *member*, and *assistant*; or specific rights such as John Smith. Every time a client tries to perform a certain interaction, the applet first checks locally to see if the user has the right access; if no, an "access denied" error message is returned; otherwise, the user action is executed. During the session, when a manager changes these access rights, the server sends these changes to the appropriate client applet.

D. Event Collision

Event collision is not a new issue. It has existed for many years in database management systems (DBMS), for example. The problem occurs when the events of two or more users which are generated at approximately the same time affect the same data and create unwanted results. For instance in an on-line presentation, one user might advance the presentation to the next slide while another is trying to annotate on the previous slide. As a result, the annotation might appear on the new slide. This situation is depicted in figure 2. It is apparent that only one of these events should have been allowed.

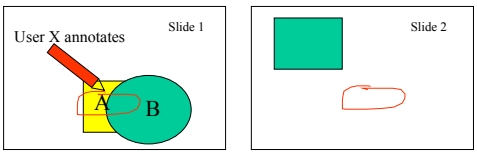

Figure 2. Event collision a)what user X intended b)what actually happened because another user changed the slide.

Event collision is usually addressed by some type of access control. Optimistic techniques, such as validation [11], are usually not suitable for these situations. In validation, all checking for collisions is done at once and after execution of operations. The assumption in optimistic approaches is that there are not many user interactions being performed at the same time. Although sometimes true, this is often not the case with telecollaboration sessions. Pessimistic techniques, which perform a certain degree of checking before execution of events, seem to be more appropriate for telecollaboration. One of the common pessimistic approaches is locking. When a client wants to interact with an applet, the applet checks with the central server whether the shared application is free. If so, it locks the application such that other clients will be denied availability. After it is finished, it releases the lock so others can use the applet.

E. Synchronization

Another important aspect of collaboration is synchronization. Typically, individual users in the same session have different processing powers and different network access in terms of network bandwidth and network latency. This creates the possibility that the state of one copy of the applet is different form the state of another copy. Lack of synchronization will lead to both temporal and application state inconsistencies. For example, it will be possible for a user with high-bandwidth access to quickly bring up an image from a server and edit it, while another user with lower bandwidth has not even finished downloading the image. Hence, it is crucial for a synchronization scheme to exist in a collaboration space. Such a synchronization scheme must support timeliness, causality, and state awareness in order to insure consistency among participants [13]. This is not a trivial task. For instance in the above example, one can implement a mechanism such that no one can start editing until all participants have downloaded the image. Obviously, this will not be very efficient in a case where every user has a 1.5 Mbps network access except for one participant who has a 28.8 kbps network access. Another approach is to have the editing process of the faster stations buffered and played back for a slower station. However, this approach will not give the users of the slower stations a fair chance to participate in the editing process.

IV. PROTOTYPE IMPLEMENTATION

In order to implement the above collaboration techniques, we developed the JETS system. JETS is a groupware toolkit, written fully in Java, that also includes some basic Java applets specifically designed for collaboration.

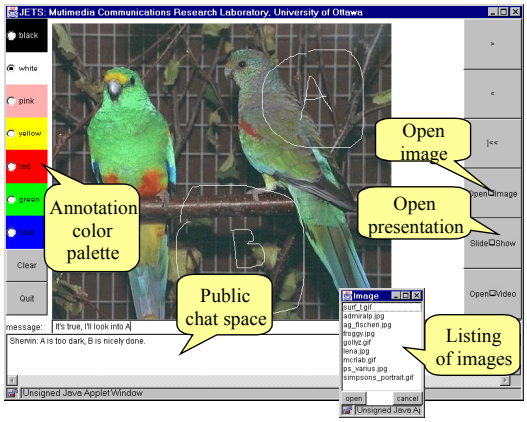

Figure 3. Whiteboard and its features.

One these applets is a multimedia whiteboard. Other than being a window for shared color drawings, the whiteboard is capable of bringing up images in JPEG or GIF format from the Web server and displaying them. The users can then annotate on these images and start a discussion. As seen in figure 3, the whiteboard is equipped with a shared chat space which can be used by participants to exchange textual massages. The whiteboard is also an on-line presentation tool since it is capable of bringing slides from the Web server and displaying them. These slides can be PowerPoint slides saved in HTML format, or simple sequences of images.
In addition, the whiteboard has the ability to play ITU-T H.263 compliant video [8]. When a user opens a video file and starts playing it, the video data is streamed down from the server to all participants, decoded in real-time, and displayed in their whiteboard.
When we first decided to write the H.263 video-decoder in Java, we were expecting slow performance of the decoder because of Java performance problems. However, we were able to obtain up to 25 frames per second for a QCIF video with the whiteboard running in the Netscape Communicator 4.04*.
A management system was also implemented to enable monitoring of the session. Each client is provided with a

* Tested on a 200 MHz Pentium with 64MB RAM running Windows NT 4.0 workstation.

session bar that displays the client's access rights for each application. A client joins a session by entering username and password into a login menu (figure 4a).

Figure 4 a) User "Pete" tries to join the session. b) The session chair sees that Pete has joined, c) access rights are indicated via colored buttons d) chairperson can deny or grant an access request sent by a user.

The *session chair*, a predetermined participant who acts as a moderator, can see who has entered the session and can extract specific information about each participant's access rights (figure 4b). The type of access for each application is indicated by colored buttons. Red for no access, yellow for view-only access, and green for view/interact access (figure 4c). A client can request for higher access rights from the chairperson and receive those rights upon chairperson's approval.

The chairperson can also change the access rights of a participant dynamically during the session and without the participant's request.

Another feature of the session management system is the ability to vote on a subject. The chairperson can permit participants to vote on a subject that has been indicated on the chat utility of the whiteboard. Users can express their vote by pressing yes or no on a vote pop-up menu. The result of the vote is reported back to the chairperson.

V. TESTING AND PERFORMANCE ANALYSIS

During testing, we had to replace the event broadcasting mechanism by a *message multicasting system* architecture where applets, upon user interaction, send predefined messages, instead of events, to other clients through the server [12]. This was due to the inconsistencies discovered in the event handling of the applet. Although many events were intercepted and sent to the other clients without any problem, some events such as typing in a TextField were not handled properly by the receiving clients. We believed this to be a problem of implementation of Java AWT by either the Web browser or the Java Development Kit (JDK) itself. A paper by Begole et al [4] confirms the existence of this problem and proposed solutions that must be considered by JavaSoft itself.

To evaluate JETS in terms of system delay and response, it was tested on both local and wide-area networks and for both ATM and regular Internet connections. A test applet was designed to measure the effective *client-to-client delay (CCD)* of the system. CCD is an indication of the average time it takes for a byte of data to travel from one client to another and includes the delays caused by all protocol levels such as the application level, the transport level, and the network level. The CCD results were compared with QoS parameters for Multimedia Desktop Collaboration (MDC) published by the Multimedia Communications Forum [9]. It was shown that on LANs and ATM WANs the JETS system has acceptable quality in terms of user interaction and human perceptions. Typical CCD values were in the 50-60 msec range [12].

VI. RELATED WORK

As mentioned earlier, there are other Java collaboration systems that have emerged. These are the NCSA Habanero, the Java Collaboration Environment (JCE) [2], and Java Applets Made Multiuser (JAMM) [5]. Among these, JAMM is a system that more closely compares with JETS in terms of its objective since it tries to achieve collaboration using Java applets rather than Java applications.

The main difference between JAMM and JETS is their collaboration enabling architectures. JAMM uses transparent collaboration. This means that the events are automatically intercepted and sent to the other copies by the collaboration engine. A developer writes a single-user applet and won't have to know that the applet will be a collaborative one. This however requires that the core JDK be modified. Hence, in order to run JAMM, one requires to download and install a modified version of the JDK. On the other hand, JETS uses the standard JDK and can run in any Java-enabled Web browser without any modifications although it requires the developer to handle the events once they are intercepted.

Even though a transparent collaboration system makes it easier for the developer to write typical applets, it doesn't always work for the case of multimedia applets. Consider the following scenario for a shared video player. Assume the video data is streamed from the server to every client. Due to the difference in access parameters such as network bandwidth and delay, different users will have different views of the video file. If someone presses the pause button in this case, simply intercepting the pause event and sending it to all applets is not enough because when other applets receive the pause event, they will not necessarily freeze on the same frame as the originating one and most likely different users will see a different frame.

To avoid this problem, one has to specify on what frame pause was pressed. In other words, additional information must be sent together with the event. Consequently, one can see that pure transparent collaboration techniques will not be sufficient in such instances. Even though a secondary system with its own communication channels can be used between the clients to handle this problem, this will create redundancy since a communication infrastructure for event passing already exists and ideally that should be enough to pass messages among all copies of the applet. Subsequently, a collaboration system must offer either an optional message passing mechanism or make the underlying

communication channels available to the developer.

VII. CONCLUDING REMARKS

We briefly described some of the main implementation issues that are encountered in telecollaboration systems and we discussed that a client-server approach is appropriate for a Java applet collaboration system. Although distributed approaches could also be used to implement many of the issues described in section 3, the inherent properties of Java applets make a client-server approach more justifiable. These properties include the primary existence of a server which is part of the system and from which the applets are downloaded, as well the security restrictions such as communication channels, which are only allowed between the applet and the server, and file manipulation limitations.

One of the main issues left open is synchronization between clients, or *inter-client synchronization*. As we saw in section 3.5, this is very important for multimedia collaborative applets. Unlike *intra-client synchronization*, which ensures synchronization between multiple media on a single client, inter-client synchronization addresses the issues of synchronizing media between multiple clients. As illustrated earlier in the video decoder example, when one client presses the pause button during the video playback, the video on all other clients must freeze on the exact same frame. Taking the pause-initiating client as the reference, this means that some mechanism must be developed to accommodate both the clients who lag behind and the clients who have already moved ahead in the video presentation. Furthermore, this mechanism must work in a real-time fashion. This argument can be extended to other multimedia applications as well. This area of Web-based collaboration requires more research.

JETS is a working example of an applet-based multimedia collaboration system. Although at its early stages of development, JETS successfully demonstrates that Java collaboration frameworks are not only possible but also can perform at acceptable levels of quality if designed correctly, despite the performance deficiencies pointed to by Java critics.

REFERENCES

[1] C. Gutwin and S. Greenberg, “Workspace Awareness for Groupware”, Proc. ACM Computer-Human Interface (CHI ’96), ACM press, New York, 1996.

[2] H. Abdel-Wahab, J. Favereau, O. Kim and P. Kabore, J. Favereau “An Internet Collaborative environment for Sharing Java Applications” IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems (FTDCS'97), Tunis, Tunisia, October 29 - 31, 1997

[3] H. P. Dommel and J. J. Aceves, “Floor Control for Multimedia Conferencing and Collaboration”, ACM Multimedia Systems, Vol. 5, No. 1, 1997, pp. 23-38.

[4] J. Begole, C. Struble and C. Shaffer, “Leveraging Java Applets: Toward Collaboration Transparency in Java”, IEEE Internet Computing, March-April 1997, pp. 57-64

[5] J. Begole, C. Struble, C. Shaffer and R. Smith, "Transparent Sharing of Java Applets: A Replicated Approach," Proceedings of the 1997 Symposium on User Interface Software and Technology (UIST'97), ACM Press, NY, 1997, pp. 55-64.

[6] J. Grudin, "Computer-Supported Cooperative Work: History and Focus", IEEE Computer, Vol. 27, No. 5, pp 19-26, May 1994.

[7] J.C. Oliveira, "TVS - A Videoconferencing System"; Master Dissertation (in Portuguese), Computer Science Department, Pontifical Catholic University of Rio de Janeiro, Brazil, August 1996

[8] K. Rijkse, "H.263: Video Coding for Low-Bit-Rate Communication", IEEE Communications Magazine, December 1996.

[9] Multimedia Communication Forum Inc., “Multimedia Communication Quality of Service”, MMCF document MMCF/95-010, Approved Rev 1.0, September 24, 1995.

[10] O. Kim, P. Kabore, J. Favereau and H. Abdel-Wahab, “Issues in Platform-Independent Support for Multimedia Desktop Conferencing and Application Sharing”, Proceedings of the Seventh IFIP Conference on High Performance Networking (HPN'97), White Plains, NY, April 18 - May 2, 1997

[11] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems* 2nd edition, Benjamin/Cummings Publishing Company, California, 1994, Chapter 8.

[12] S. Shirmohammadi and N. D. Georganas, “JETS: a Java-Enabled Telecollaboration System”, Proc. IEEE ICMCS, IEEE Computer Society, Los Alamitos, Calif., 1997, pp. 541-547.

[13] W. Robbins and N. D. Georganas, “Shared Media Space Coordination: Mixed Mode Synchrony in Collaborative Multimedia Environments”, Proc. IEEE ICMCS, IEEE Computer Society, Los Alamitos, Calif., 1997, pp. 466-473.