Java Multimedia Telecollaboration

Jauvane C. de Oliveira
National Laboratory for Scientific Computation, Brazil
Military Institute of Engineering, Brazil

Mojtaba Hosseini, Shervin Shirmohammadi, François Malric, Saeid Nourian, Abdulmotaleb El Saddik, and Nicolas D. Georganas
University of Ottawa, Canada

Using Java-based tools in multimedia collaborative environments accessed over the Internet can increase an application's client base. Most operating systems support Java, and its "compile once–run everywhere" architecture is easy to maintain and update. The Java-based tools presented here let users share Internet resources, including resources originally designed for single use.

Multimedia collaborative environments let geographically distributed users share multimedia resources. The Internet and World Wide Web offer a globally accessible network and easy user interface that can extend these systems to greater numbers of users. To more effectively use the Web for collaboration, developers have implemented multimedia systems for various platforms. For example, the Lawrence Berkeley National Laboratory/University of California, Berkeley, videoconferencing tool (VIC)[1] comes in several versions compiled for various flavors of Unix (including Linux), Windows, and so on.

If we had a single build for all platforms, we would only need to update that code. Java's "compile once–run everywhere" architecture offers one possible solution. Many operating systems support Java, and any of them could benefit from Java-compliant applications. Java also uses applets, executable code that runs in a Web browser. Because applets are downloaded from a Web server with the Web page, the administrator need only update the code on the server to ensure that every client runs the same version. No other language is so easy to maintain. With Java, developers don't have to worry about making new systems compatible with previous versions.

Java provides acceptable performance in some rather complicated collaborative multimedia system prototypes.[2–4] (See also the "Related Work on Java-Based Collaboration" sidebar.) A developer who adheres to PersonalJava, a subset of Java, can count on an even greater client base, because all Java-enabled operating systems are also PersonalJava enabled. Many operating systems that can't handle the complete Java set—for example, personal digital assistant (PDA) operating systems such as Windows CE—have a PersonalJava implementation available.

At the Multimedia Communications Research Laboratory at the University of Ottawa, we've developed a comprehensive set of Java- and PersonalJava-compliant multimedia systems for collaboration. These systems include a video decoder, shared whiteboard, and shared Web browser. We've also implemented a prototype system for sharing any existing Java applets and applications, even those designed for single users. Several of these tools are based on the Java-enabled telecollaboration system (JETS) infrastructure, which we will describe in greater detail. Performance evaluations show that JETS can support collaboration sessions with hundreds of concurrent users. In addition, we've developed collaborative virtual environments based on Java/Java3D.

jStreaming: H.263 video decoder

jStreaming (http://jStreaming.com/) decodes standard ITU-T H.263[5] video streams. Because it's Java-compliant, jStreaming can run as an applet in any Java-enabled Web browser or as a Java application in any Java-enabled operating system.

jStreaming's core complies with Java development kit 1.0.2 (the first widely available release of Java) and thus also complies with more recent Java implementations. Such compatibility means that even older versions of browsers (Netscape Navigator 3.0 and Microsoft Internet Explorer 3.02, for example) can deploy jStreaming.

jStreaming also complies with PersonalJava, a subset of JDK 1.1 that addresses limited devices such as PDAs. We've achieved a 10-frames-per-second (fps) playback rate, quarter common intermediate format (QCIF), with a Compaq iPAQ 3670 PocketPC and have received reports of jStreaming achieving over 144 fps in Windows. In 1998, we achieved 34 fps on a 333-MHz Pentium II. Native code achieves about 55 to 60 fps in the same system.

jStreaming can stream video from a multithreaded video server (also written in Java) using a simple protocol with sliding window flow control, or from a Web server using HTTP. Figure 1 shows jStreaming running in a PocketPC and in a desktop system.

1070-986X/03/$17.00 © 2003 IEEE

Published by the IEEE Computer Society

Related Work on Java-Based Collaboration

Although many Java-based collaboration systems exist, none offer management or moderation features similar to those in Java-enabled telecollaboration system (JETS) or Jasmine.

Kuhmünch developed a Java remote-control tool, which lets users control and synchronize distributed Java applications and applets.[1] As in JETS, Java applets and applications are sharable only when used with the system application programming interface (API). JavaSoft's Java shared data toolkit (JSDT) is also an API-based framework.

Habanero is another Java-based approach that supports the development of collaborative environments.[2] The Habanero framework helps developers create shared applications, either from scratch or by modifying existing single-user applications to integrate the new collaborative functionality. Instead of using applets, which can be embedded in almost every browser, Habanero uses *happlets,* which require the user to download and install a proprietary browser.

The National Institute of Standards and Technology (NIST) developed the Java Collaborative Environment (JCE), based on the collaborative abstract window toolkit (C-AWT), an extended version of the Java-AWT.[3] This approach requires replacing AWT components with the corresponding C-AWT components.

Because these approaches all propose an API, creating a collaborative application requires modifying an application's source code and reimplementing it, or designing and implementing a new application.

Another approach is to use an X Window system protocol such as SharedX[4] or technologies based on NetMeeting.[5] These approaches, however, aren't amenable to all groups of users. NetMeeting runs only on Windows, and to use SharedX, you must install an X server and X clients. Moreover, NetMeeting doesn't provide the moderation capabilities that jStreaming and JETS do.

References

1. C. Kuhmünch, T. Fuhrmann, and G. Schöppe, "Java Teachware: The Java Remote Control Tool and Its Applications," *Proc. ED-Media/ED-Telecom 98*, Assoc. for the Advancement of Computing in Education (AACE), 1998, pp. 70-75.
2. A. Chabert et al., "Java Object Sharing in Habanero," *Comm ACM*, vol. 41, no. 6, June 1998, pp. 69-76.
3. H. Abdel-Wahab et al., "An Internet Collaborative Environment for Sharing Java Applications," *Proc. IEEE Computer Society Workshop Future Trends of Distributed Computing Systems* (FTDCS), IEEE CS Press, 1997, pp. 112-117.
4. D. Garfinkel, B. Welti, and T. Yip, "HP SharedX: A Tool for Real-Time Collaboration," *Hewlett-Packard J.*, Apr. 1994, pp. 23-26.
5. Microsoft Corporation, "NetMeeting Resource Kit," http://www.microsoft.com/windows/NetMeeting/Corp/reskit/, 1999, current 22 May 2003.

Figure 1. A sample jStreaming session in which clients use a PocketPC and a desktop for video playback.

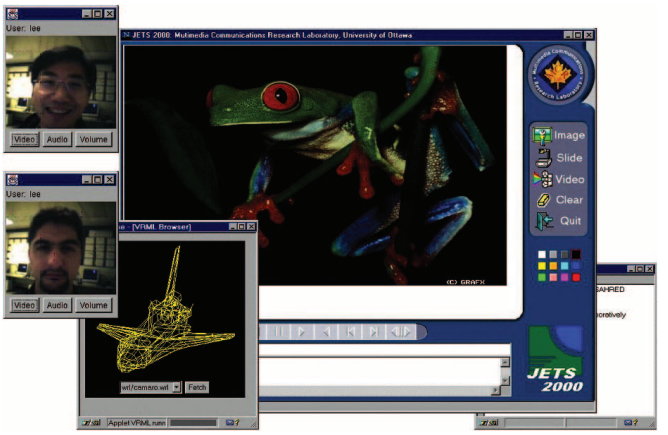


Figure 2. A sample JETS session with shared applets and audio- and videoconferencing.

jStreaming provides a high-level application programming interface (API), so it can be bundled into other prototypes.

JETS: Framework for sharing applets and applications

JETS (http://www.mcrlab.uottawa.ca/jets) is a client-server framework for sharing Java applets and applications.[6–8] Because JETS uses the core Java packages, users don't need to install additional Java classes on their systems, and can access the system and share applets using a Java-enabled browser. JETS 2000, the most recent version, also offers videoconferencing through the Java media framework (JMF).

Figure 2 shows a screen shot of a sample JETS session. JETS includes many utilities that enable multimedia viewing and sharing.

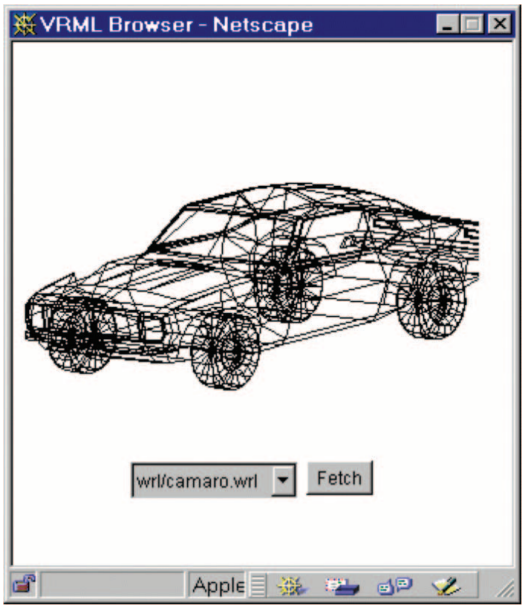


Figure 3. VRML browser. As users interact with the 3D object, JETS reflects all activity on all session participants' screens.

Whiteboard

JETS' main interface consists of a control panel, chatting dialogue box, and shared whiteboard area. The whiteboard is an interactive space where clients in a virtual session can share slides, text, video, drawings, and so on. Users can also annotate images or start a discussion. A built-in locking mechanism prevents more than one user from modifying the same object at the same time.

The simplest way to interact on the whiteboard is with text in the chat area. All group members with input access to the whiteboard will see the message originator's name followed by the message.

Users can also draw on the whiteboard. To draw, a user simply chooses a color from the color template, presses the left mouse button, and drags the mouse. Pressing the "clear" button erases the drawings.

A client can paste an archived picture onto the whiteboard by clicking "image" with the mouse button and choosing a file in the image file dialog box. The picture will instantly appear on all members' whiteboards. Any member with full access can comment or draw on the picture.

A client can also start an archived slide show by selecting "slide" with the mouse button and choosing a file in the slide show dialog box. Again, a member with full access can comment or draw on the slide show or go to the next or previous slide using the VCR buttons.

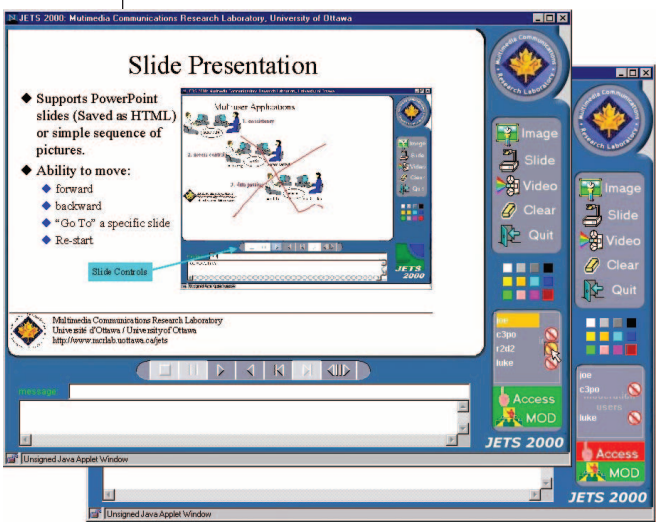


Figure 4. Access control in JETS. Clients request access to the shared whiteboard from the session moderator by clicking the "access" button.

Shared video presentation

Users can play ITU-T H.263-compliant video on the whiteboard using the jStreaming API. When a user plays a video file, jStreaming streams the video data to all participants, decodes it in real time (processor permitting), and displays it on their whiteboards.

VRML viewer

Another applet is a shared 3D viewer for Virtual Reality Modeling Language files that permits real-time collaborative interaction with simple VRML objects. The applet brings VRML 1.0 files from the server and displays them. Users can collaboratively interact with the 3D objects, with all rotations, translations, and zooming reflected on all participants' screens. Figure 3 shows the shared VRML browsing interface.

Videoconferencing and recording

The Java video conference recorder tool provides services for videoconferencing, recording sessions, and playing recorded sessions. J-VCR records sessions in Synchronous Multimedia Integration Language format—a World-Wide Web Consortium (W3C) standard. As a result, any SMIL-player (RealNetwork's RealPlayer, for example) can play back the recorded session.[9]

Session management

JETS implements a management system for monitoring sessions. A session client logs in as moderator by clicking the "MOD" button, which initiates session management. Once session management is established, only the moderator has the right to access the shared whiteboard. Other session clients must request access permission by clicking the "access" button. The moderator either grants or rejects the request. If the moderator approves the request, the client becomes a session participant and can access the shared whiteboard and annotate files on it.

The moderator can revoke a client's access privileges at any time. The client can, however, request access permission again. Figure 4 shows

two JETS windows. In the front window, the green "access" icon indicates that the client can interact with the whiteboard. The red "access" button in the back window indicates that the user can see updates to the whiteboard but can't interact with it.

Architecture

From a developer's viewpoint, JETS is a set of APIs for building shared resources, providing built-in consistency, access control, and data passing capabilities.

Figure 5 shows the JETS multithreaded server. When a user joins the session, the main server launches a subserver for that user. Subservers are responsible for processing only the update messages or requests from their clients. When the subserver receives an update message, it sends it to all other session participants, creating fast system response but using more resources due to subserver threads. However, floor control usually dictates that only one client at a time can control and interact with an application, and most threads simply wait. JETS uses transmission control protocol (TCP)/Internet protocol (IP) and user datagram protocol (UDP)/IP sockets for client–server communication.

In Figure 5, client 1's interactions with applet A are reflected to data server 1, which runs as part of server A for application A. Next, data server 1 relays client 1's actions to the clients listed in a client list on server A. Finally, client 2's applet A receives the actions and reflects them on that user's screen.

Performance evaluation

JETS is a real-time tool in the sense that its update response time, in network environments that support real-time applications, is within the acceptable parameters of human quality of service for desktop collaboration. But as with any TCP-based multiuser system, there's an upper bound to how many simultaneous users can be on the system before those parameters are violated. This limit depends on system resources, such as processing and graphics power, memory, and network bandwidth and delay, as well as the design of the system's communication.

Depending on the quality of service desired, the application-level end-to-end delay between two users should be less than 1,000 ms, with 200 ms recommended for tightly synchronized tasks.[9,10] However, these numbers are valid only if we use the shared application with some type of media that provides a sense of presence, such as video and audio. If audio or video or both are present, users have a sense of awareness of each other, and the shared application must respond within a time period that maintains this awareness.

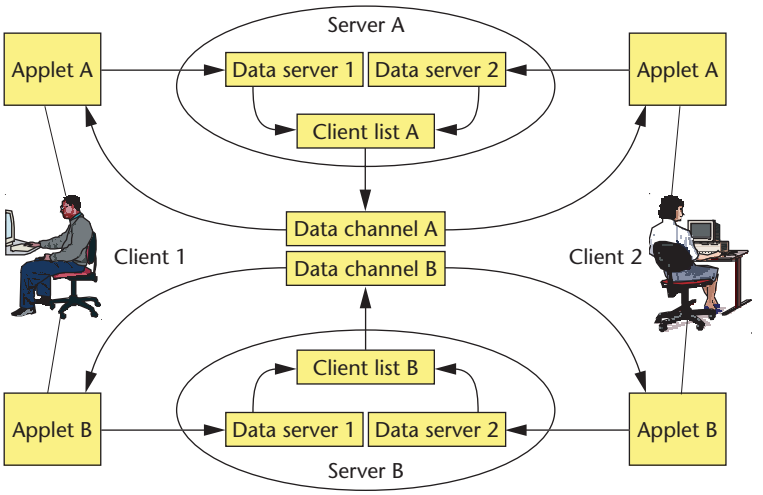


Figure 5. Client-server communication in JETS. The main server assigns a subserver to participants as they join the session. Subservers process and update messages from their assigned user only.

For example, say three engineers are collaboratively designing a bridge in a live session. One engineer highlights a section of the bridge and says, "I think this part should be redesigned." If they are using real-time audioconferencing (end-to-end audio delay of 100 ms), the shared application's delay must comply with the above numbers if the other two engineers are to receive the audio message and the event update quickly enough to maintain the session's real-time quality. This is usually the case in controlled IP environments such as local networks or corporate intranets.

For typical Internet connections, where audio and video delays aren't controllable, or in the absence of audio or video, strict delay parameters make little sense because users have no time-wise perception of one another. Thus, a user who receives an update message has no way of knowing when the action occurred. Even a delay of five seconds or more might be acceptable, depending on the nature of the application.

Our performance evaluations assume a controllable environment that both requires and supports real-time interaction.

Parameters of interest. *Client-to-client delay* (CCD) is the most common parameter for measuring a collaborative application's quality. CCD tries to measure the average time it takes for an

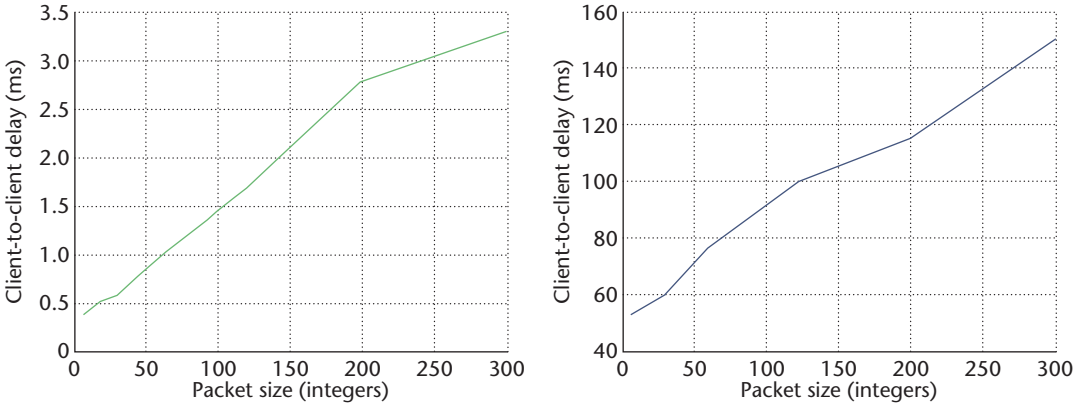


Figure 6. Client-to-client delay results for packet-based communication in JETS: (a) over a 100-Mbps Ethernet; (b) over a 28.8-Kbps modem line.

update message to reach other users. It includes all layers between the two clients, including application-, transport-, networking-, and physical-layer delays. At the application level, however, CCD only measures the time it takes for a sender to send or a receiver to receive the update. It doesn't include the delay caused by the application's actions with the update because this delay is application dependent. For example, if a user opens an image in a whiteboard, we measure how long the "open image" message takes to reach all clients. We don't measure how long it takes the receivers to download the image and display it on their screens because the JETS server doesn't control those delays.

In addition, the server processing time per packet increases with the number of simultaneous users. This is because of the one-to-one TCP connection-oriented nature of the system: the server must send the update to the clients one by one. This *server processing delay* (SPD) adds to the system's overall end-to-end delay, and we must consider it when calculating the maximum number of users the system will support.

Another interesting parameter is the *floor control delay* (FCD), the average time it takes for the system to grant or to deny access to a user. FCD measures a system's intuitiveness. A system with a smaller FCD is easier to use because the user receives feedback from the system faster and in a more real-time fashion.

Testing and results. We tested the CCD, SPD, and FCD of JETS over both a 100-Mbps local area network (LAN) and a 28.8-Kbps telephone modem. All machines were running their typical network and operating system background processes during testing.[11]

To test CCD, we had a sender applet send an event to a receiver applet. On receiving the event, the receiver applet extracted all necessary data from the packet, reassembled the event, and sent the event back to the sender. The sender likewise extracted the data, reassembled the event, sent it back to the receiver, and so on. This was repeated for 10 minutes. Figure 6a shows the test results when performed over a 100-Mbps Ethernet.

We measured packet size by the number of integers sent per packet. Typical packet sizes ranged from three integers (draw a point with given colors) to eight (draw a line from point A to B with a given color) and larger. Although it's unlikely that a 300-integer message would be sent in one packet, we extended our test to that limit to see the effect of very large update messages. Figure 6b shows the CCD test performed over a 28.8-Kbps modem.

In the SPD test, the sender applet flooded the server with event updates. The receivers (ranging from 1 to 45) then calculated the average delay for adjacent packets. As Figure 7a shows, the delay increased with the increasing number of users. Figure 7b shows the same test performed for updates of various sizes. Because of floor control and moderation, no more than one client at a time could send events to the server, a typical scenario in collaborative applications.

Because the server spends equal processing time per packet per client, delay increases linearly as the server sends messages to more clients.

In the FCD test, a client constantly requested control and then released it on receipt for a given

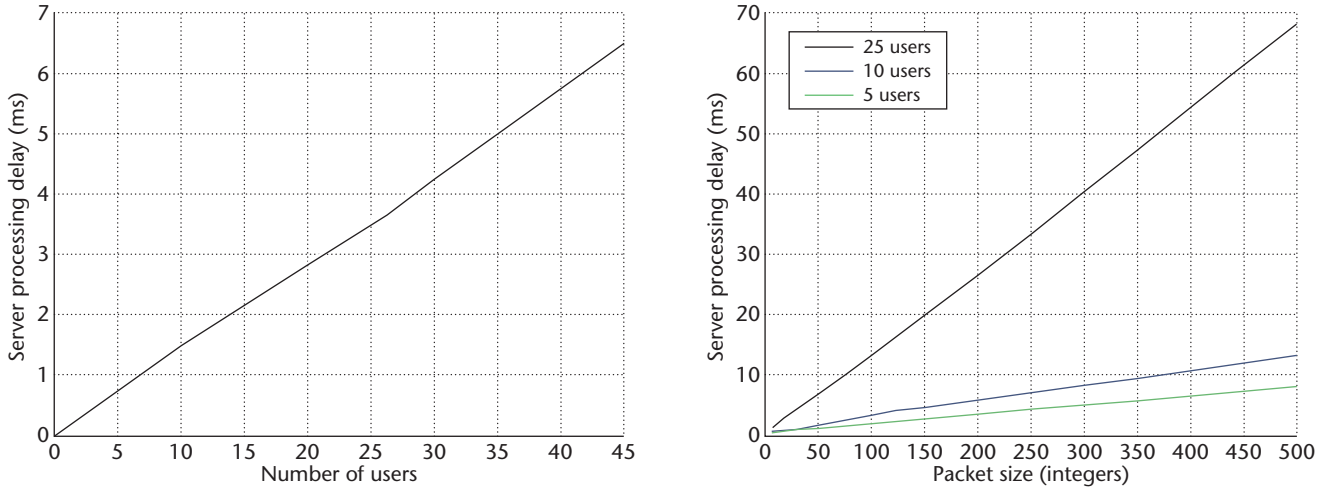


Figure 7. JETS server processing delay: (a) with an increasing number of users and (b) with packet sizes.

amount of time. The average FCD was less than 5 ms, which affirms the intuitiveness of the system's floor-control mechanism.

Analysis. As mentioned previously, the recommended overall end-to-end delay is less than 1,000 ms, with less than 200 ms required for closely coupled collaboration. This is true for CCD, SPD, and the application-dependent on-screen rendering and display delay. The rendering delay (RD) isn't constant and depends on the hardware, operating system, and platform used.

From the CCD and SPD tests, we can approximate the overall delay as delay = CCD + SPD + RD; and, as Figure 7a shows, SPD $\approx 0.142 \times N$, where N is the number of users. Hence,

$$\text{delay} \approx \text{CCD} + 0.142 \times N + \text{RD},$$

which roughly represents the delay from the time a client's interaction generates a typical event to the time the interaction appears on other clients' screens.

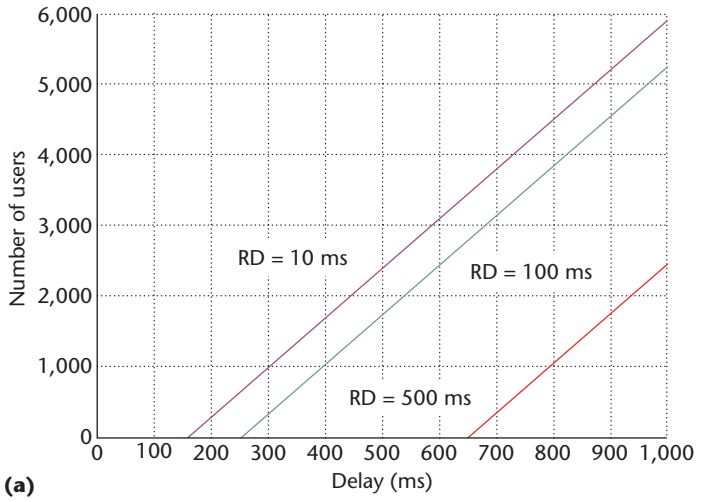
Figure 8a (next page) shows the achievable number of users based on the expected overall delay for different rendering delays. Figure 8b shows the number of users, but focuses on tightly synchronized tasks (delay is less than 200 ms). Finally, Figure 8c shows the number of users JETS can support over a 28.8-Kbps modem.

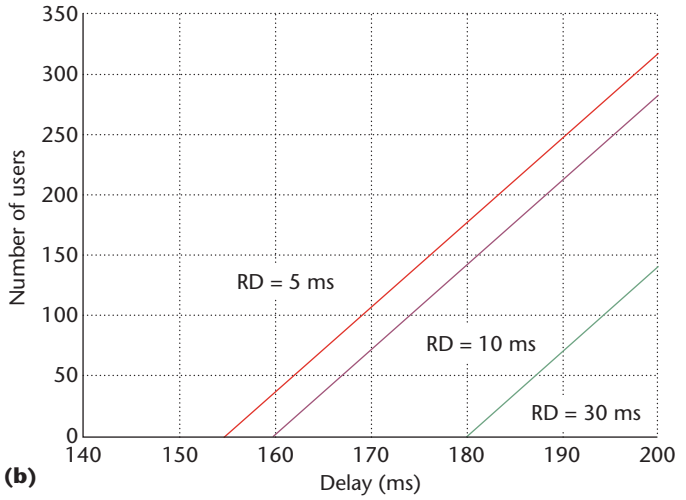
The graphs in Figure 8 show that the system can support many users. However, although the plots suggest that JETS can support thousands of users, the actual number is smaller. The system's linear behavior diminishes with more users: Server performance decreases substantially as we approach the maximum allowable socket connections on the equipment, and the underlying physical network slows as the number of users increases. Nevertheless, JETS' underlying communication module can support collaboration sessions of hundreds of users, resources permitting.

Jasmine: Environment for collaboration-unaware applets

The Java application sharing in multiuser interactive environments (Jasmine)[11] prototype exploits the many useful resources and objects that have been developed as Java applets and applications and are available on the Internet. It lets users share these applets and applications in real time without modifying the code. Jasmine targets collaboration-unaware applications and applets, enabling the use of almost all single-user applets and applications in a collaborative way. Jasmine's architecture can help people collaborate in the growing number of computing environments in which Java applications and applets run over IP.

Figure 9a shows the Jasmine framework wrapping around an applet to be shared. The framework listens to the events occurring in an applet's GUI, capturing both Java abstract window toolkit (AWT)-based and Swing-based events. After capturing an event, Jasmine sends it to the communication module, which sends the event to all other session participants (Figure 9b).





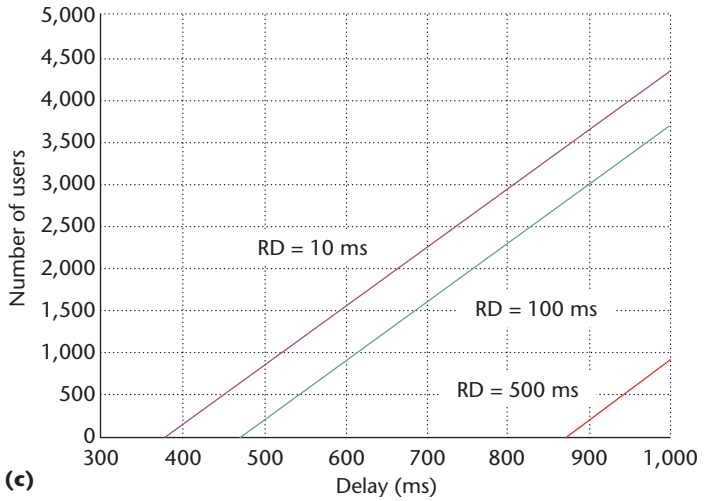


Figure 8. Expected overall delays for different numbers of users for: (a) different rendering delays, (b) tightly synchronized tasks (delay is less than 200 ms), and (c) a 28.8-Kbps modem.

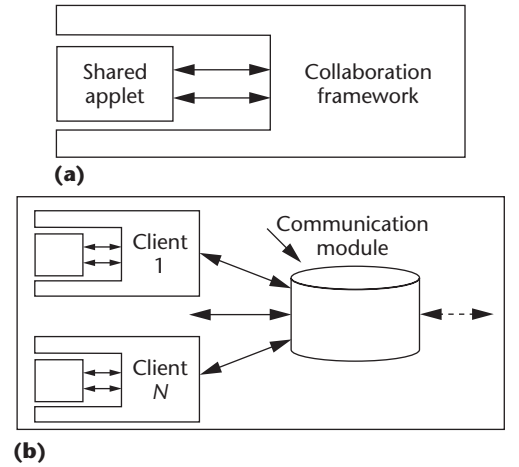


Figure 9. Jasmine architecture. (a) The Jasmine framework listens for events in an applet's GUI. (b) After capturing an event, Jasmine sends it to the communication module, which sends it to the other session participants.

Figure 10 shows a sample Jasmine session with arbitrary applets and resources users have brought in from the Internet. Because users can bring any Java applet or application into the session, Jasmine has an unlimited extendibility, with each resource enhancing the system dynamically.

Jasmine also lets users dynamically manage the collaborative session and grant access. This feature is useful in environments requiring tight control. In teletraining applications, for example, an instructor might use this feature to prevent students from changing the presentation without permission.

Moderation and permission components manage access to Jasmine's shared resources. An instructor logs into a session with a special password and presses the "moderation" button to enable session moderation. After this button has been pushed, any participant wishing to interact with the shared applications must request permission by pressing the "permission" button. The moderator receives users' requests and can either grant or deny them. A green light on a participant's permission button indicates that permission has been granted, and the user can interact with the application. The moderator can cut off any participant by pressing the "cut" button next to the participant's name.

The Jasmine client is responsible for capturing events, sending events to the server, receiving events from the server, and reconstructing events locally. The client, which is a Java application,

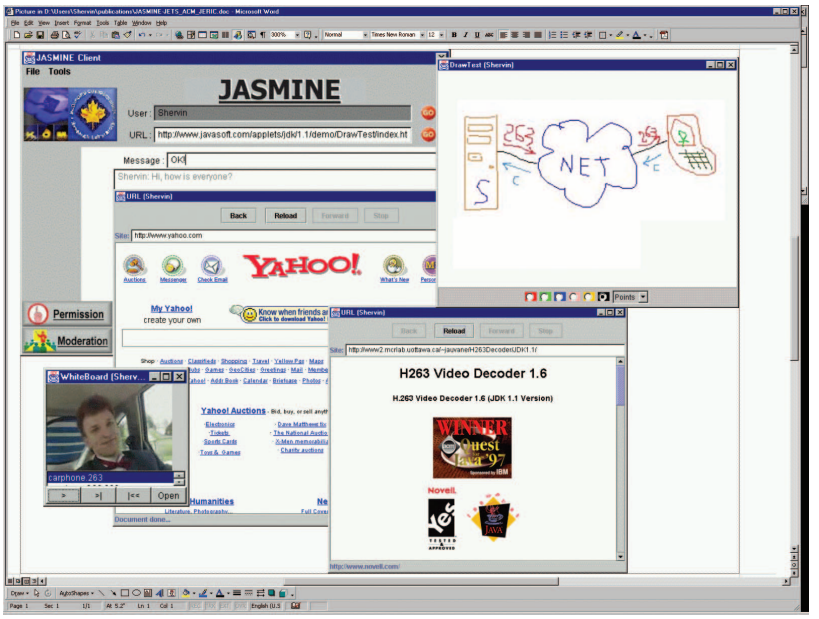


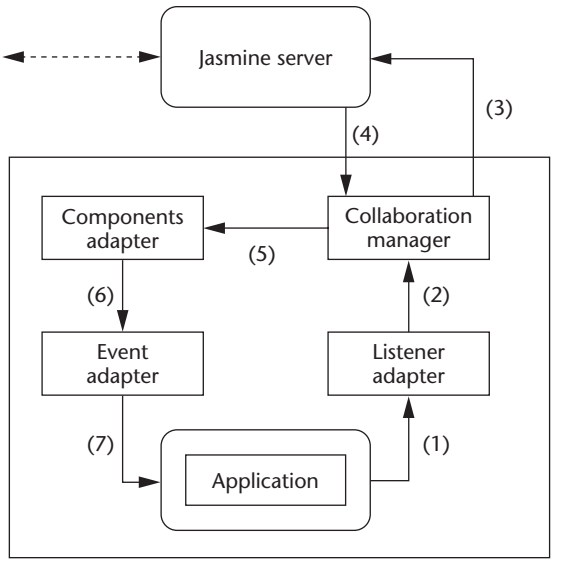
Figure 10. Main Jasmine application. The session includes any number of Java applets and resources brought in from the Internet.

has four important components:

- The *collaboration manager* is the main component on the client side. It oversees communication between clients and the server and provides a GUI.
- The *listener adapter* implements several AWT listeners. It converts events into remote events and forwards them to the collaboration manager.
- The *component adapter* maintains a list of references from all applications and applets to their GUI components. This list is created in the same order on each client, so components have the same reference numbers on all clients.
- The *event adapter* works opposite the listener adapter, converting remote events to local events and applying them to corresponding components.

Figure 11 presents a dataflow diagram of Jasmine's client-side architecture. The system consists of two main data paths:

- *Path 1 (labeled 1–3).* The listener adapter catches Java events occurring in a Java application. If the event is local, the listener adapter converts it into a remote event and forwards it to the collaboration manager, which sends it to the Jasmine server.



- *Path 2 (labeled 4–7).* The collaboration manager forwards the remote event to the component adapter, which gets information about the event source and sends this information with the event to the event adapter. The event adapter converts the remote event to a local AWT event and dispatches the event to the corresponding component.

Because the Jasmine server uses the same communication techniques as the JETS server, its performance characteristics are similar to JETS' in terms of network delay and other parameters.

Figure 11. Jasmine has two main data paths: from the listener adapter to the Jasmine server, and from the server to the application.

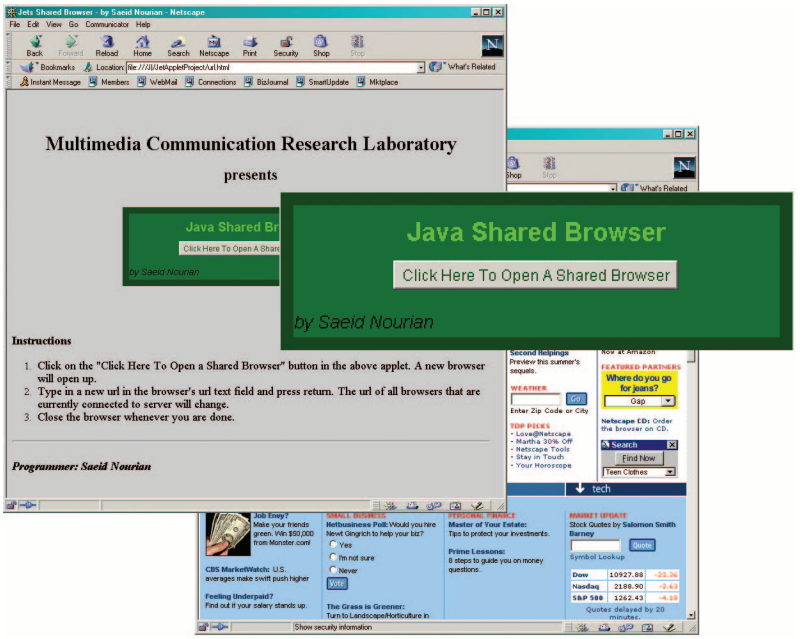


Figure 12. Jasber's interface. Jasber is a client applet that lets users share a Netscape browser.

Jasber: Framework for sharing Web browsers

The Java shared browser (Jasber) lets users share a Netscape browser. Figure 12 shows the Jasber interface. Jasber is a client applet that communicates with a JETS server. Loading the applet establishes a client–server connection. The user can then open a shared browser by clicking on the appropriate button. Any change made in the browser's HTML content (changing the URL, for example) is detected by Jasber, which updates the HTML content of the other shared browsers by forwarding the new URL through the server.

Jasber ensures consistency among shared browsers by:

- Detecting when a user types a URL into the shared browser's address field to load a Web page, and forwarding the URL to the server. The server forwards the address to all Jasber clients in the session. Consequently, all shared browsers connected to the server will load the new Web page.
- Detecting when a user's action (such as clicking on a hyperlink) causes a Web page address to change in one or more frames belonging to a shared Web page, and updating the modified frames in all other shared browsers.

The Jasber applet, and its corresponding HTML page, can be stored in a single, known Web server. When a client browser loads the HTML page from this server, Jasber starts up as an applet and opens a communication channel to the appropriate JETS server. Users interested in a shared browsing session simply visit the known Web server and launch the shared browser.

As a pure-Java applet, Jasber works on any platform with a standard Java virtual machine (JVM) implementation. Jasber uses standard Java libraries (such as java.util) to store the list of URLs gathered from the current browser page. JETS libraries, which provide a simple yet effective framework for Jasber, enable client–server communication.

Jasber also uses two essential libraries from Netscape:

- It uses netscape.javascript to read and change Netscape browsers' content.
- It uses netscape.security to negotiate security permissions with the users.

These Netscape libraries are compatible only with Netscape Communicator browsers.

Figure 13 shows the three main modules of the Jasber architecture:

- URLApp communicates with the server using JETS classes.
- WatchBrowser, a subclass of Java thread, con-

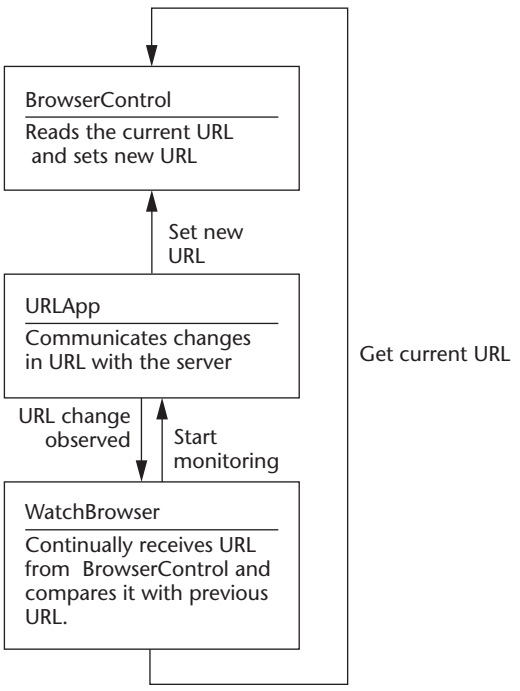


Figure 13. Jasber architecture. Three main modules enable communication with the server, URL monitoring, and reads and writes to the browser.

tinually monitors the shared browser's current URL.

- BrowserControl directly reads and writes to the shared browser. It uses Netscape security classes to meet user permission requirements for reading and changing browser properties.

Java and virtual environments

One area in which Java is perhaps not considered a viable option is distributed virtual environments. The Java3D API lets developers quickly and easily create 3D applications and applets entirely in Java.

We developed a 3D application to provide remote industrial training in a virtual setting. In the application, two geographically distant users engage in a session in which a trainer teaches a trainee how to install various hardware components. Java provides the core technology, rendering, user interface, communication, and multimedia integration. Figure 14 shows the application's architecture and the component Java technologies.

While Java3D provides a flexible and powerful 3D-rendering API, we've used the Java native interface (JNI) to connect our applications with several input devices such as Immersion's CyberGlove (http://www.immersion.com), Ascension Technology's 6DoF miniBird (http://www.ascension-tech.com), and SensAble Technologies' Phantom haptic device (http://www.sensable.com). We use Java media framework (JMF) to integrate video and audio capabilities with the application.

We've used these Java technologies to develop sophisticated distributed virtual environment applications involving immersive and stereoscopic displays, multiple input devices, and mixed-media environments (such as video inside a 3D world).

Figure 15 shows a training application using a CyberGlove to track the hand's movements and a Phantom for haptic feedback (both connected via JNI). A JMF transmits the trainer video and renders it inside a Java3D scene.

Java's main advantage in distributed virtual environments is its platform independence, as was demonstrated during our recent move from PC-based solutions to SGI-based ONYX machines. We easily transferred our Windows-based applications to run on the Irix platform without having to recompile the Java code. Hence we can run our distributed virtual environment applications on both platforms. Furthermore, Java3D's flexible API allows for the 3D content to be displayed in immersive multiple-screen environments, head-mounted display (HMD) as well as on a regular monitor.

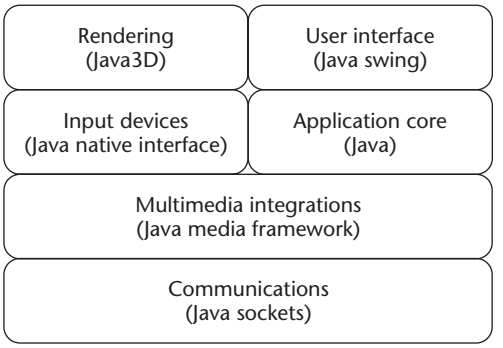


Figure 14. Java in a virtual environment architecture.

Figure 15. Remote training using multiple input devices, video, and 3D media.

In addition to Java's interoperability and other features, Java3D's high-level programming interface allows quick development of 3D applications without requiring computer graphics expertise. The ability to include synthetic media (3D graphics) in Java applications and applets opens up yet another avenue in multimedia collaboration in which Java can have a significant impact.

Ongoing and future work

The telecollaboration systems described in this article have received much attention from

both academia and industry, including many citations and license agreements.

Both jStreaming and JETS have been certified in Sun's 100-percent Pure Java program and Novell's "Yes" logo program. An earlier version of jStreaming won third prize in the ACM/IBM Quest for Java 97, and an earlier version of JETS won first prize in the ACM/IBM Quest for Java 98. Several companies worldwide have licensed jStreaming and JETS. jStreaming also led to the Ottawa Centre for Research and Innovation (OCRI) 2001 Futures Award: Student Entrepreneur of the Year.

Current research includes techniques for latecomer support and client synchronization, and multiparty videoconferences in 3D virtual environments that includes haptic/touch feedback as a new medium of communication. The scalability and feasibility of such media-rich applications over best-effort internetworks is thus of special interest. **MM**

Acknowledgments

We acknowledge the financial assistance of the Brazilian Ministry of Education Agency's CAPES scholarship, the PCI/LNCC fellowship program, the Ontario Research and Development Challenge Fund, the TeleLearning Network of Centres of Excellence, the Communication and Information Technology Ontario (CITO), and Newbridge Networks/Alcatel.

References

1. S. McCane and V. Jacobson, "vic: A Flexible Framework for Packet Video," *Proc. ACM Multimedia*, ACM Press, 1995, pp. 511–522.
2. J.C. de Oliveira, S. Shirmohammadi, and N.D. Georganas, "Collaborative Virtual Environments Standards: A Performance Evaluation," *Proc. IEEE Distributed Interactive Simulation and Real-Time Applications Workshop* (DIS-RT), IEEE Press, 1999, pp. 14-21.
3. J.C. de Oliveira, X. Shen, and N.D. Georganas, "Collaborative Virtual Environment for Industrial Training and e-Commerce," *Virtual Reality Technologies for Future Telecommunications Systems*, A. Pakstas and R. Komiya, eds., Willey, 2002, pp. 79-102.
4. J.C. de Oliveira et al., "Virtual Theater for Industrial Training: A Collaborative Virtual Environment," *Proc. 4th World Multiconf. Circuits, Systems, Comm., and Computers* (CSCC), World Scientific Eng. Soc., 2000, pp. 294-299.
5. Int'l Telecomm. Union, "Video Coding for Low Bitrate Communication," ITU-T Recommendation H.263, Feb. 1998.
6. S. Shirmohammadi and N.D. Georganas, "JETS: Java-Enabled TeleCollaboration System," *Proc. IEEE Conf. Multimedia Computing and Systems*, IEEE Press, 1997, pp. 541-547.
7. S. Shirmohammadi, J.C. Oliveira, and N.D. Georganas, "Applet-Based Telecollaboration: A Network-Centric Approach," *IEEE Multimedia*, Apr.–June 1998, pp. 64–73.
8. Multimedia Comm. Research Lab., *JETS2000 User Manual*, 2001.
9. S. Shirmohammadi, L. Ding, and N.D. Georganas, "An Approach for Recording Multimedia Collaborative Sessions: Design and Implementation," *J. Multimedia Tools and Applications*, vol. 19, Kluwer Academic, no. 2, 2003, pp. 135-154.
10. Multimedia Comm. Forum, "Multimedia Communication Quality of Service," MMCF document MMCF/95-010, 1995.
11. S. Shirmohammadi et al., "Jasmine: A Java Tool for Multimedia Collaboration on the Internet," *J. Multimedia Tools and Applications*, Kluwer Academic, vol. 19, no. 1, 2003, pp. 5-28.

Jauvane C. de Oliveira is a researcher in the Computer Science Department at the National Laboratory for Scientific Computation (Brazil), and a professor in the Systems and Computer Engineering Department at the Military Institute of Engineering (Brazil). His research interests include collaborative virtual environments, multimedia communications, virtual reality, distributed systems, ubiquitous computing, and computer networking. He has a BS in computer science from the Federal University of Ceará, Brazil, an MS in computer science from the Pontifical Catholic University of Rio de Janeiro, and a PhD in electrical and computer engineering from the School of Information Technology and Engineering at the University of Ottawa, Canada. He is a member of the IEEE, the IEEE Computer Society, the ACM, and the SBC (Brazilian Computing Society).

Mojtaba Hosseini is a PhD candidate at the University of Ottawa's School of Information Technology and Engineering. His research interests are collaborative virtual environments and 3D

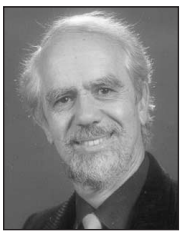
videoconferencing applications. He has an MS from the University of Ottawa.

Shervin Shirmohammadi is a software architect at OEone Corp., Canada, and a part-time professor at the University of Ottawa, Canada. His research interests include collaborative virtual environments, multimedia communications, and telecommunications software. He has an MS and a PhD in electrical engineering from the University of Ottawa.

François Malric is a research engineer and system administrator at the Discover Laboratory. His research interests include Java technologies, haptics, and immersive virtual reality. He has a BS in computer science from the University of Ottawa.

Saeid Nourian is a graduate student in computer science at the University of Ottawa. His research interests include virtual reality, specifically the design and implementation of physical laws on top of Java3D. He has a bachelor's degree in software engineering from the University of Ottawa.

Abdulmotaleb El Saddik is an assistant professor in the School of Information Technology and Engineering, University of Ottawa, and an associate editor of the *ACM Journal of Educational Resources in Computing* (JERIC). His research interests include configurable and adaptable component-based development of instructional multimedia and large-scale learning systems with particular interest in XML, Web services, and learning object metadata. He has an MS and a PhD in electrical engineering and information technology from Darmstadt University of Technology, Germany. He is a member of the ACM, the IEEE, and the ACS (Arab Computer Society).



Nicolas D. Georganas is Distinguished University Professor and Canada Research Chair in information technology at the School of Information Technology and Engineering, University of Ottawa. His research interests include multimedia communications, collaborative virtual environments, Web telecollaboration applications, intelligent Internet sensors and appliances, and telehaptics. He has an Dipl. Ing. in electrical engineering from the National Technical University of Athens, Greece, and a PhD in electrical engineering (summa cum laude) from the University of Ottawa. He is a fellow of the IEEE and the Engineering Institute of Canada.

Readers may contact J. Oliveira at the National Laboratory for Scientific Computation, Department of Computer Science, C.P. 56065, Rio de Janeiro, RJ, 22290-970, Brazil; email@jauvane.com.