Flexible Collaboration Transparency

James "Bo" Begole   *Clifford A. Shaffer*
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
{begolej,shaffer}@cs.vt.edu

ABSTRACT

This paper presents two distinct contributions: First, we present a critique of collaboration transparency as it is currently implemented in contrast to collaboration-aware implementations. We find conventional collaboration-transparency systems lacking in terms of efficient use of network resources and support for key groupware principles: concurrent work, relaxed WYSIWIS, and group awareness. Second, we examine the causes of these deficiencies, and then present an alternative implementation approach based on an object-oriented replicated architecture where selected single-user interface objects are dynamically replaced by multi-user extensions. The replacement is transparent to the single-user application and its developer. As an instance of this approach, we described its incorporation into a Java-based collaboration-transparency system, called JAMM.

KEYWORDS: Computer-supported cooperative work, groupware, collaboration transparency, multi-user interface components, workspace awareness, Java.

INTRODUCTION

There are two general approaches to providing computer support for synchronous collaboration. One is the sharing of legacy single-user applications. This is referred to as **collaboration transparency,** because the sharing is provided by a mechanism that is unknown, or transparent, to the application and its developers. The second approach, **collaboration awareness,** is to design an application specifically to support cooperative work.

There is an ever-growing set of legacy single-user applications that coworkers may like to use in support of their collaborative tasks. The primary benefit of collaboration-transparency systems is that, by supporting the shared use of such legacy applications, developers make no extra effort whatsoever for their applications to be shared. In contrast, collaboration-aware applications are developed *ad hoc* or with a groupware toolkit, such as Bellcore's Rendezvous [11], the University of Calgary's GroupKit [20], and the University of Michigan's Collaboratory Builder's Environment (CBE) [16]. In this case, a software developer makes some additional effort beyond that of developing an otherwise functionally equivalent single-user application.

In terms of development cost, then, collaboration transparency has the advantage. However, there are also differences in the quality of support for collaboration that these types of systems provide. A fundamental difference is in the type of tasks they support. Collaboration-transparency systems primarily support tasks that extend from individual to collaborative performance, such as text editing. In addition to supporting such individual-to-collaborative types of tasks, collaboration-aware applications can also support inherently collaborative tasks, such as voting, chatting, group decision making, and multi-player games. For example, although single-user adventure games exist, sharing one via collaboration transparency does not make it equivalent to a Multi-User Domain (MUD) where the actions of distinct individuals are important.

When a sufficiently powerful single-user application is available to support a task, however, it makes sense to consider collaboration transparency. Indeed, given the lower development cost of collaboration transparency, is there any reason to develop a collaboration-aware application for such tasks? This is the motivating question for the work reported here.

We present our work in two parts. First is a critique of conventional collaboration-transparency systems with respect to their support of key groupware principles and use of network resources, along with a discussion of the implementation reasons for various deficiencies. We approach the investigation by contrasting group support capabilities seen in conventional collaboration-transparency systems to collaboration-aware implementations. Researchers of collaboration awareness have made scattered criticisms of collaboration transparency, but to our knowledge, no thorough critique of conventional collaboration transparency exists. In the second part, we describe an approach to collaboration transparency that provides many features previously seen only in collaboration-aware applications. Our approach is based on a replicated architecture under which selected single-user interface objects are dynamically replaced with multi-user extensions. Transparency is maintained, because the original application source code is unmodified. We illustrate the approach with examples of its incorporation in a replicated, collaboration-transparency system, called JAMM (Java Applets Made Multiuser) [2, 3].

CRITIQUE OF COLLABORATION TRANSPARENCY

Several collaboration-transparency systems now exist, particularly for the X Window System. Examples include Old Dominion University's XTV [1], Hewlett Packard's SharedX [7] and Sun Microsystem's ShowMe SharedApp [26]. Recently, Microsoft has made NetMeeting, a collaboration-transparency system for the Windows platform, freely available [17]. Because of its free availability and widely-used platform, NetMeeting is helping popularize collaboration transparency among casual computer users.

We refer to these and other currently available collaboration-transparency systems as **conventional,** because they share the same basic implementation and resulting limitations described in this section, as opposed to the **flexible** collaboration-transparency system that we discuss in the second half of this work. Prior research in collaboration transparency has focused on the feasibility of implementation approaches, not on the usefulness of these systems for actual collaborative work. Based on our analysis and a review of the scattered complaints of other researchers, we found that conventional collaboration-transparency systems do not adequately support collaboration. This section enumerates these shortcomings and explores their causes in conventional collaboration-transparency implementations.

Lack of Support for Groupware Principles

Research in the field of computer-supported cooperative work (CSCW) has identified many principles that effective collaborative software should incorporate. This section describes the deficiencies of conventional collaboration transparency in key groupware principles: no concurrent work, strict WYSIWIS, and limited group awareness.

No Concurrent Work

Under conventional collaboration-transparency systems, only one participant at a time can work in the shared application. The controlling participant is said to *hold the floor.* Floor control policies range from **explicit,** where participants explicitly grab and release control of the application, to **implicit,** where the collaboration-transparency system automatically passes the floor among collaborators as they use the application. In all cases, however, only one collaborator at a time may actually use the application.

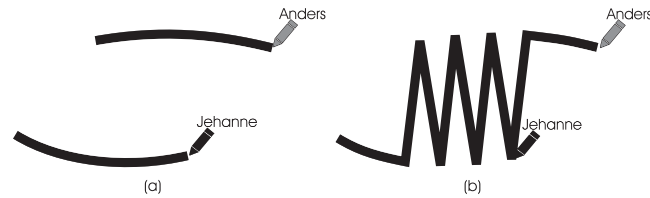

Figure 1: (a) Desired and (b) actual output caused by event collisions

The degree to which collaborators work closely together versus independently is referred to as **tight** versus **loose coupling** [4]. Floor control imposes a tightly-coupled mode of collaboration, because while one participant is in control, the others may only observe the work. However, group work is not always so tightly coupled and many researchers have cited this restriction as a serious problem with conventional collaboration transparency [18, 19, 22]. In contrast, many collaboration-aware applications allow concurrent independent work.

Cause: Application-level Locking Collaboration-transparency systems must maintain the intended behavior of the shared application. If input events from multiple participants are simultaneously applied to the shared application, event streams for non-atomic events such as mouse drags may become interleaved, and conflicts between collaborators can arise.

For example, in Figure 1, Anders and Jehanne are sharing a drawing application, and both attempt simultaneously to draw separate freehand curves by dragging their mouse cursors. However, the location of the previous mouse-drag event for Anders is different from that of Jehanne. When the next mouse-drag events are delivered, they conflict because the event for Jehanne uses the context from the event for Anders. Figure 1a shows what was intended. Figure 1b shows the desired output corrupted by the mouse-drag event collisions. The application has drawn a line between the alternating mouse positions of both Jehanne and Anders, instead of between the sequential positions of Jehanne's mouse, followed by sequential positions of Anders' mouse. This is an unexpected and undesirable result.

To avoid such conflicts, conventional collaboration-transparency systems use floor control, restricting input to only one participant at a time. Floor control is essentially locking at the coarsest possible granularity – the whole application. In contrast, a collaboration-aware application may also use locking, but at a finer granularity permitting safe, simultaneous manipulation of different portions of the shared data. Such fine-grained locking supports a more loosely-coupled collaboration than floor control allows.

Strict WYSIWIS

In conventional collaboration-transparency systems, participants have the same view of the shared application in a manner referred to as strict **What You See Is What I See (WYSIWIS),** where the participants see exactly the same view of the shared application at the same time [25]. This also contributes to the imposition of tightly-coupled collaboration, since participants must view the same portion of the shared data simultaneously. In contrast, collaboration-aware applications often support multiple paradigms of collaboration by relaxing WYSIWIS so that participants can simultaneously view different portions of shared data [10, 24].

Cause: Centralized Architecture The architecture used by all conventional collaboration-transparency systems is a **centralized display broadcasting** architecture. In this approach, participants receive only a graphical depiction of a central instance of the shared application. In contrast, fully **replicated** architectures maintain consistent copies of the shared data at each participant's host. Only the information needed to update the state of each copy is distributed, potentially using less network bandwidth than display broadcasting. Fully- and semi-replicated architectures are commonly supported by collaboration-aware groupware toolkits, such as Groupkit [20] and DistView [18], and are often seen in *ad hoc* collaboration-aware applications such as multi-user games like Doom and Diablo.

A centralized architecture ensures that the state of the shared application is consistent among the participants, because there is only one instance of the application. Unfortunately, since the application is shared in a strict WYSIWIS manner, centralized display broadcasting is a principle cause of conventional collaboration transparency's poor support of fundamental groupware principles.

Limited Group Awareness

Schuckmann et al. point out that conventional collaboration transparency does not provide detailed information concerning participants' actions and locations [22]. Typically, conventional collaboration-transparency systems provide awareness information only by means of **telepointers,** representations of remote users' cursor positions. In some collaboration-transparency systems, like Microsoft NetMeeting, one pointer is shared among all participants and is controlled by only one at a time. In others, like Sun's ShowMe SharedApp, each participant has a unique telepointer.

Cause: Strict WYSIWIS In fact, telepointers do provide sufficient information about participants' locations for a collaboration-transparency system, since participants share a strict WYSIWIS view of the application. However, G. Smith as well as Greenberg and Roseman demonstrate that in environments that allow users to have simultaneously independent views of the shared application, called **location-relaxed WYSIWIS,** simple telepointers are inadequate [23, 8]. To provide more detailed location information in addition to telepointers, other multi-user interface elements may be used to indicate remote participants' locations. Gutwin and Greenberg have demonstrated, for example, that multi-user radar views enhance the usability of collaborative applications for some collaborative tasks [9].

Network Usage

In addition to lack of support for groupware principles, conventional collaboration-transparency systems compare poorly to collaboration-aware applications in terms of efficient use of network resources. Prakash et al. point out that conventional collaboration-transparency systems use more network bandwidth than collaboration-aware applications [18].

Cause: Centralized Architecture The primary cause of the higher network demand for conventional collaboration transparency is the centralized display broadcasting architecture used by all conventional collaboration-transparency systems. The lower network demand of a replicated approach is apparent when we note that user inputs are transmitted in both replicated and centralized architectures, but the centralized approach must also distribute graphical display information.

Additionally, a replicated architecture allows the local copy of a shared application to respond more quickly to local inputs than a centralized approach. In the centralized approach each user input must travel from the originator to the central application, which reacts, potentially generating new display information which is broadcast to all participants. In a replicated approach, the local replica may act on an input first, before the input is distributed to remote replicas, providing faster local application response than in the centralized case.

Cause: Lack of Knowledge of Application Semantics Another reason that a conventional collaboration-transparency system uses higher network bandwidth is lack of knowledge of the applications' semantics, which prevents the system from optimizing network communications. Since it cannot determine how the application will respond to a given user input event, a conventional collaboration-transparency system must send *all* inputs to the application even though some might be safely omitted. For some applications, it might be acceptable to send no mouse position updates, only mouse button press and

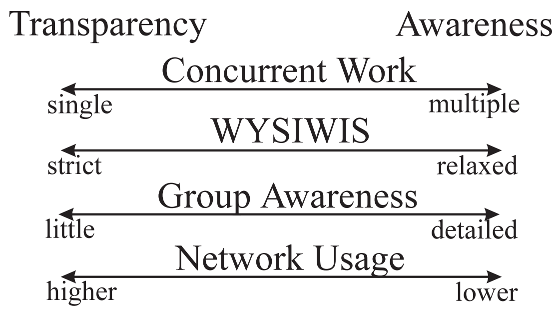

Figure 2: Four dimensions along which conventional collaboration-transparency systems and collaboration-aware applications differ.

release updates. However, since the state of some other application may depend on mouse position information, the conventional collaboration-transparency system forwards all position updates for all applications.

Developers of collaboration-aware applications, on the other hand, can take advantage of their knowledge of the application's semantics, transmitting only the information needed to maintain state consistency among the replicas and provide acceptable performance [5].

Summary: Awareness versus Transparency

To summarize, there are at least four distinct dimensions along which transparency and awareness systems currently differ in their support for collaboration: concurrency, WYSIWIS, group awareness and network usage. Figure 2 illustrates where the two types of systems lie along these dimensions. It is important to note that while conventional collaboration-transparency systems are clustered at the left end of each dimension, collaboration-aware applications range up and down the scales; a particular collaboration-aware application may be no better than conventional collaboration transparency in one or all dimensions.

FLEXIBLE COLLABORATION TRANSPARENCY

Despite the problems previously enumerated, collaboration transparency remains attractive, because of the low development cost of sharing legacy applications. We propose a way to provide support of groupware principles to collaboration transparency. Our approach is to combine a replicated architecture with dynamic replacement of particular single-user interface objects with multi-user extensions. This section describes the replicated architecture and object replacement technique and the advantages they afford. We illustrate the approach by describing its incorporation into an instance of a replicated-architecture collaboration-transparency system called JAMM (Java Applets Made Multiuser), described in prior work [3].

Replicated Architecture

Our approach uses a replicated architecture, whereas conventional collaboration-transparency systems use centralized architectures. In prior work, we found that, current technology can support a replicated architecture for transparent sharing of many, though not all, single-user applications [3, 15]. In addition to allowing lower network bandwidth usage, replication is essential to provide support of concurrent work and relaxed WYSIWIS, as described later in this section.

Dynamic Object Replacement

Today, the user interface of an object-oriented application is typically constructed by combining graphical user interface (GUI) objects. As useful interface objects become popular, application platforms provide them as standard components. Examples of such are the classic Windows, Icons, Menus, Pointers, and Scrollbars (WIMPS). Many compound interface components are also commonly available, such as scrollable lists, editable text areas, and dialog windows.

Polymorphism In our approach to flexible collaboration transparency, particular standard single-user interface components are replaced by multi-user equivalents at the time the application is initially shared. This scheme depends on the polymorphism of an object-oriented platform, such as Smalltalk and Java. Polymorphism allows an instance of one class to exist in place of another class, provided the replacement has an object interface that is signature-equivalent to the original. Thus, an instance of a single-user interface class may be substituted by an instance of a polymorphically-equivalent multi-user class. Since the application source code is not modified, such substitution is considered transparent to the shared application.

Semantic Knowledge For those interface objects that it replaces, the flexible collaboration-transparency system has knowledge of the semantics of input events that occur within that object. With this information, the system can optimize its use of network resources by minimizing the size and number of update messages distributed to replicas. This semantic knowledge also allows a flexible collaboration-transparency system to provide relaxed WYSIWIS, implicit floor control, concurrent work, and fast local application response. The remainder of this section describes our implementation of this approach and the use of semantic knowledge to realize these advantages.

Flexible JAMM

We have incorporated our approach into a replicated-architecture collaboration-transparency system, called JAMM [3]. Our current version of JAMM is implemented with the Java Development Kit 1.1.5.

Figure 3 shows a single-user text editing application, called Stylepad, which is included as a sample application in the Java Foundation Classes/Swing library [14]. This application contains a scrollable panel that in turn embeds a text editor. In Figure 4 the same application is now being shared among three participants via JAMM. The scrollable panel object has been replaced by a multi-user *radar* panel.

JAMM Target Platform We have constrained the set of applications that can be shared to those that are developed strictly using the Java Foundation Classes/Swing user interface library [14]. JAMM's application migration mechanism further limits its set of shareable applications. JAMM uses an **image-copy** scheme based on JOS to transfer the current state of a shared application to new-comers of the session. For an application to be copied via JOS, however, the developer must choose to make all application classes **serializable**. Although in most cases serializability is not difficult to implement, a developer may have no need of it and therefore not include that capability.

Collaboration-transparency systems generally allow the sharing of any application written for a particular platform (e.g., X or Windows). Due to the constraints outlined here, JAMM's target platform is not the entire set of Java applications, but the subset of serializable, Swing-based Java 1.1 applications.

Information Points As a replicated collaboration-transparency system, JAMM intercepts and distributes low-level input events. In addition, the Java 1.1 event delegation model allows JAMM to register itself as a listener to higher-level user interface actions, such as scrollbar position changes. This registration is transparent to the original single-user application. Thus, JAMM is notified of the semantic result of certain low-level input events, allowing it to optimize network usage and provide location-relaxed WYSIWIS, as described later in this section.

Dynamic Object Replacement In Java, polymorphism is provided such that an instance of a subclass may exist in a reference to its superclass. Therefore, an instance of a multi-user subclass may replace an instance of a standard single-user superclass.

In JAMM this replacement is provided by Java Object Serialization (JOS). JOS is part of the core Java class library in Version 1.1 and provides the means to copy and transfer an object. For each object reachable from the initial target object, JOS provides a mechanism to inspect and replace it with an instance of a polymorphically-equivalent class. Thus, the replacement of one object with another is transparent to the application.

Flexible JAMM replaces instances of two Swing classes at the time an application is shared. Each instance of `com.sun.java.swing.JScrollPane` is replaced by a multi-user radar pane, providing several benefits to collaboration: location-relaxed WYSIWIS, detailed group awareness, and immediate local scroll position updates. JAMM also substitutes each instance of `com.sun.java.swing.text.PlainDocument`, which is the default data model of a Swing text component, with a multi-user document, providing significant groupware benefits: independent concurrent work, and immediate local response to edits.

We selected these two classes for replacement, because we considered the development cost to be reasonable and the benefits to collaboration to be considerable. We next describe the use of semantic knowledge of input events to the multi-user replacements to provide key groupware benefits: relaxed WYSIWIS, detailed awareness information, flexible floor control, concurrent work, and fast local application response.

Relaxed WYSIWIS In addition to having a lower network bandwidth requirement, replication and semantic knowledge allows a flexible collaboration-transparency system to relax WYSIWIS by providing participants with independent views of the shared application. In contrast to the traditional replicated approach, in some cases the system does not distribute a given input event at all, but only applies it locally.

For example, the result of an input event in a scrollbar is to move the position of the scrollbar and in turn a viewport. In this case, a flexible collaboration-transparency system need not distribute the mouse event to all replicas, but only apply it locally. Hence, the flexible collaboration-transparency system provides location-relaxed WYSIWIS via independent viewport positions.

Note that it is possible for an application to include the view position in its state. In that case, the replicas might become inconsistent. A flexible collaboration-transparency system should allow the user to turn off view relaxation, falling back to the traditional replicated approach of distributing all input events, thus preventing replica inconsistency but losing the advantage of relaxed WYSIWIS.

Awareness Information Allowing location-relaxed WYSIWIS means that replicas of a component may have independent state. To allow participants to coordinate their work, it is necessary to provide information to them about each others' view state.

Continuing the previous example, after the scroll position is changed locally, the system should send a message to remote participants indicating the new local viewport position. In response to a viewport change notification,

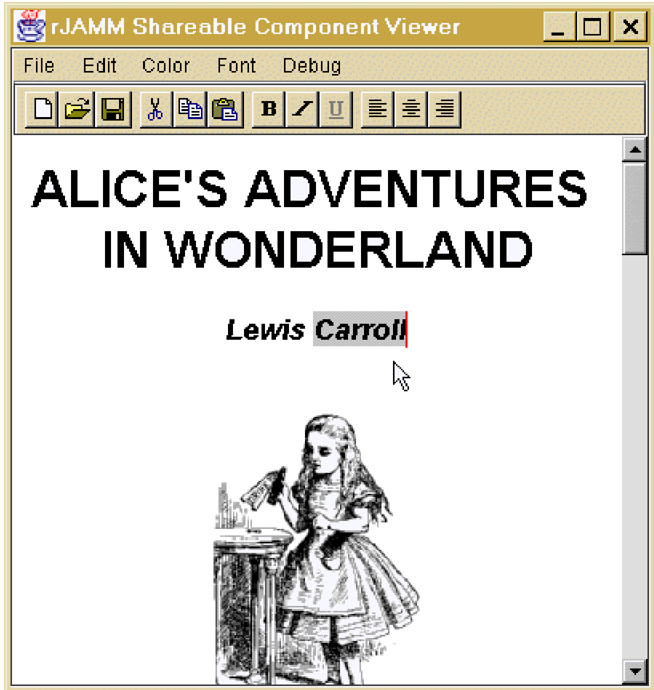

Figure 3: A single-user text editing application, Stylepad, that contains a scrollable panel as a main interface component.

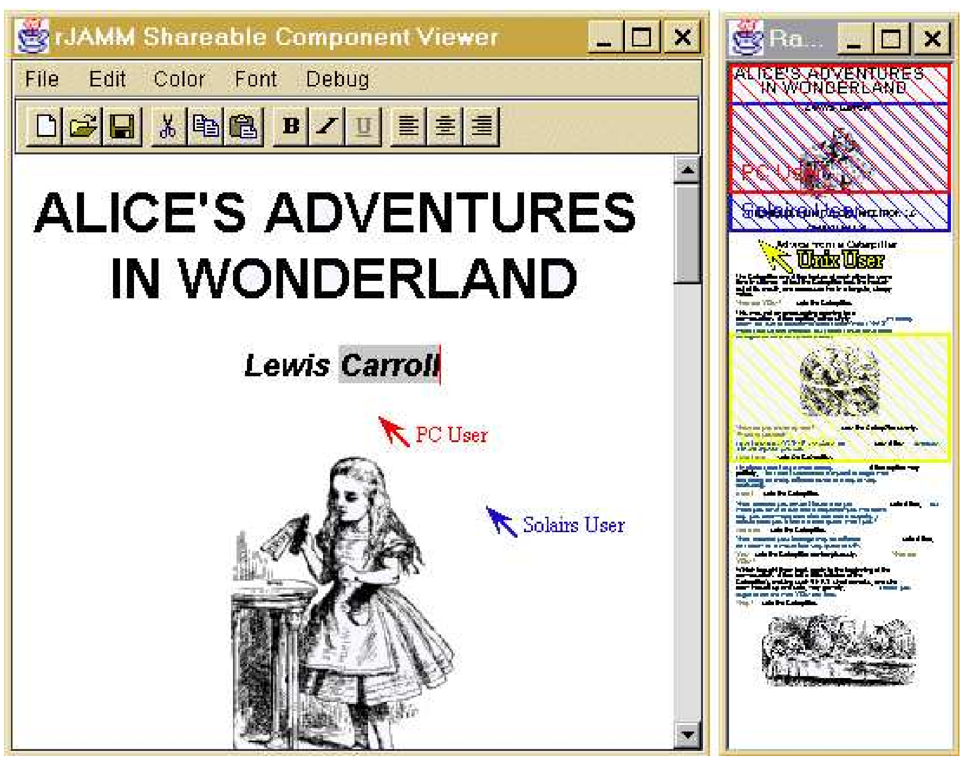

Figure 4: Stylepad shared via JAMM. The single-user scrollable panel has been replaced by a multi-user radar panel. The window on the right displays a miniature overview, or *radar view*, of the scrollable panel's content (mixed text and images, in this document). A uniquely colored and shaded rectangle indicates each participant's viewport location in the panel. Three participant viewports are shown in this example: *PC User* and *Solaris User* overlap near the top of the document while *Unix User* is near the center.

Figure 5: JAMM's input control panel indicating that control is currently held by the collaborator named Janice. Input control may be explicitly requested by pressing the "Take Control" button, or implicitly when a participant presses a mouse button in the shared application.

a remote replica will update its location-awareness interface elements, such as the viewport indicators in the radar view used by JAMM (see Figure 4). Therefore, in addition to telepointers, JAMM provides group awareness information by indicating each participant's viewport position in the radar view.

Explicit and Implicit Floor Control To avoid potential conflicts between non-atomic events as described previously, JAMM uses a floor control mechanism for some, but not all, interface components. Where floor control is used, JAMM provides mechanisms for both explicit and implicit floor requests. Implicit control provides seamless floor passing as work flows among the participants of a collaboration.

Figure 5 shows the user interface to JAMM's floor control mechanism. Control is implicitly granted to the first participant to begin a potentially non-atomic series of inputs (i.e., mouse drags) by pressing a mouse button. That participant holds control until he or she releases the mouse button and for a short time (5 seconds) after to allow the same user to quickly regain control if attempted. Control may also be explicitly requested by pressing the "Take Control" button. If the current control holder checks the "Keep Control" check-box, no one else may gain control. Otherwise, control will be granted to the next participant who implicitly or explicitly requests it.

Concurrent Work Although implicit floor control facilitates control passing, it still allows only one participant at a time to work in the shared application. However, floor control locking is not required for inputs to all components. In the case of an input in a text-editing component (i.e., `com.sun.java.swing.JTextArea`), JAMM does not impose floor control. All participants may simultaneously edit the text.

To provide this concurrent editing capability, JAMM replaces the original text data object with an extension that uses serialized transactions to update replicas. For a key or mouse event generated in a text area, JAMM does not broadcast the event, but applies it locally only. If the document is changed as a result, which is usually the case, JAMM distributes a message describing the change to all replicas. The algorithm JAMM uses is an optimistic transaction approach similar to that described by Ellis and Gibbs, using no locking yet avoiding conflicts between edits that occur as close as one character apart [6]. A replica updates its copy of the document by transforming an incoming remote edit to be consistent with its own set of local edits.

Fast Local Response JAMM does not forward all user input events; some are applied only locally. As described in the previous subsection, key and mouse events that occur within a text area are not broadcast, but are only applied locally. Similarly, inputs that occur within a scrollbar of a radar pane are also only applied locally. In both cases, any resulting state change is sent some time after the input event. Thus, the local replica of the shared application responds immediately to the input and remote replicas are updated asynchronously.

FUTURE WORK

We are currently working on the following issues.

Alternative to Object Replacement

JAMM dynamically replaces certain single-user objects with multi-user equivalents that are extensions of the original's class. There are two potential problems with this approach, although we have not yet encountered either: post-share replacement and subclasses of targets.

JAMM replaces the target objects via a mechanism provided by JOS at the point when the application is initially shared. JOS handles updating all references to the original object with references to the replacement. Unfortunately, no such mechanism is available after this initial point of sharing. Therefore, if the shared application later creates a new instance of a replaceable object, such as a scrollable panel, JAMM is unable to replace it with a multi-user extension. The other potential problem would arise if the application developer has subclassed a JAMM-replacement target class. Since the application presumably depends on the behavior of the subclass, it should not be replaced. In both of these cases, JAMM falls back to the sharing behavior of conventional collaboration-transparency systems, losing the advantages of flexible collaboration transparency.

To address these two issues, we intend to explore an alternative to the full object-replacement approach. Many classes of the Swing library expose enough information to provide multi-user capabilities without replacing the object. For example, the text editing components notify registered listeners of each text insertion and deletion, making it possible to use the Ellis-Gibbs concurrent-editing algorithm [6]. By attaching state-change listeners to target objects, we can provide multi-user capabilities without replacing the target objects, thus avoiding the cited drawbacks of object replacement.

Proxied Externalities

In our prototype version of JAMM, we implemented a technique to allow the semi-replicated sharing of sources of input such as files, sockets, and system time. We call such objects **externalities,** because they rely on access to resources external to the portable Java environment. The new version of JAMM does not have that support, and we hope to restore this in a future version.

Integration with a Collaborative Work Environment

Collaborative work environments, such as Teamrooms [21] and CBE [16], allow participants to work together in an integrated environment. However, only collaboration-aware applications developed for that particular environment may be used collaboratively; users can add single-user applications to the environment, but these may only be used individually. We plan to extend a collaborative work environment, Sieve [12], to include JAMM, thus providing the capability to share single-user applications integrated with a collaborative work environment.

CONCLUSIONS

Collaboration transparency provides a powerful capability, extending single-user applications to collaborative use. Collaboration-transparency systems primarily support the types of collaborative tasks that extend from individual to collaborative performance and for which a useful single-user application already exists. However, application developers may also consider using a collaboration-transparency system as a simple groupware toolkit. For example, a simple chess interface could quickly become a two-player game when shared via collaboration transparency. Unfortunately, conventional collaboration-transparency implementations impose an inflexible style of collaboration and do not adequately support key groupware principles: concurrent work, relaxed WYSIWIS, detailed group awareness, and efficient network usage.

We have presented an approach to flexible collaboration transparency that supports these groupware principles. Our approach is based on an object-oriented replicated architecture where selected single-user interface objects are dynamically and transparently replaced by multi-user extensions. As an example of this approach, we described its incorporation into a Java-based collaboration-transparency system, called JAMM.

This technique does not alleviate all of the limits of collaboration transparency. In addition to supporting the groupware principles enumerated in this paper, a collaboration-aware application may provide further support of cooperative work. For example, it may include participant roles and access policies.

We have found, however, that it is possible to bring collaboration transparency closer to the advantages afforded by collaboration awareness. This capability enhances the collaborative use of the growing body of legacy single-user applications.

JAMM is available for download at http://simon.cs.vt.-edu/jamm [13].

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the National Science Foundation under grant DGE-9553458. We are grateful to our collaborators on the previous version of JAMM, Craig A. Struble of Virginia Tech and Randall B. Smith of Sun Microsystems Laboratories. We also thank Mary Beth Rosson of Virginia Tech for discussions on this work and computer-supported cooperative work in general.

REFERENCES

1. H. Abdel-Wahab and M. Feit. XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration. In *Proceedings of IEEE TriComm*, pages 159–167, 1991.

2. J. B. Begole, C. A. Struble, and C. A. Shaffer. Leveraging Java Applets: Toward Collaboration Transparency in Java. *IEEE Internet Computing*, 1(2):57–64, Mar./Apr. 1997.

3. J. B. Begole, C. A. Struble, C. A. Shaffer, and R. B. Smith. Transparent Sharing of Java Applets: A Replicated Approach. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST-97)*, pages 55–64, New York, Oct.14–17 1997. ACM Press.

4. P. Dewan and R. Choudhary. Flexible User Interface Coupling in a Collaborative System. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, Group Use of Computing, pages 41–48, 1991.

5. P. Dourish. Consistency Guarantees: Exploiting Application Semantics for Consistency Management in a Collaboration Toolkit. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 268–277, New York, Nov.16–20 1996. ACM Press.

6. C. A. Ellis and S. J. Gibbs. Concurrency Control in Group Systems. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 18(2):399–407, June 1989.

7. D. Garfinkel, B. Welti, and T. Yip. HP SharedX: A Tool for Real-Time Collaboration. *HP Journal*, pages 23–26, Apr. 1994.

8. S. Greenberg and M. Roseman. GroupWeb: A Groupware Web Browser. In *Proceedings of the ACM 1996 Conference on Computer Supported Work, Video Program*, page 7, New York, Nov.16–20 1996. ACM Press.

9. C. Gutwin and S. Greenberg. Effects of Awareness Support on Groupware Usability. In *Proceedings of the 1998 ACM SIGCHI Conference on Human Factors in Computer Systems*, New York, Apr.18–23 1998. ACM Press.

10. C. Gutwin, M. Roseman, and S. Greenberg. A Usability Study of Awareness Widgets in a Shared Workspace Groupware System. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 258–267, New York, Nov.16–20 1996. ACM Press.

11. R. D. Hill, T. Brinck, S. L. Rohall, J. F. Patterson, and W. Wilner. The Rendezvous Architecture and Language for Constructing Multiuser Applications. *ACM Transactions on Computer-Human Interaction*, 1(2):81–125, 1994.

12. P. Isenhour, J. B. Begole, W. S. Heagy, and C. A. Shaffer. Sieve: A Java-Based Collaborative Visualization Environment. In *IEEE Visualization '97 Late Breaking Hot Topics Proceeding*, pages 13–16, Oct.22–24 1997.

13. Java Applets Made Multiuser (JAMM), Mar. 23, 1998. <http://simon.cs.vt.edu/jamm>.

14. Java Foundation Classes, Mar. 23, 1998. <http://-java.sun.com/products/jfc>.

15. J. C. Lauwers, T. A. Joseph, K. A. Lantz, and A. L. Romanow. Replicated Architectures for Shared Window Systems: A Critique. In *OIS90*, Computer Mediated Work Environments, pages 249–260. ACM Press, 1990.

16. J. H. Lee, A. Prakash, T. Jaeger, and G. Wu. Supporting Multi-User, Multi-Applet Workspaces in CBE. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 344–353, New York, Nov.16–20 1996. ACM Press.

17. Microsoft NetMeeting, Mar. 23, 1998. <http://-www.microsoft.com/netmeeting/>.

18. A. Prakash and H. S. Shim. DistView: Support for Building Efficient Collaborative Applications using Replicated Active Objects. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, Collaborative Editing and Reviewing, pages 153–164, 1994.

19. W. Reinhard, J. Schweitzer, G. Volksen, and M. Weber. CSCW Tools: Concepts and Architectures. *Computer*, 27(5):28–36, May 1994.

20. M. Roseman and S. Greenberg. Building Real Time Groupware with GroupKit, A Groupware Toolkit. *ACM Transactions on Computer Human Interaction (TOCHI)*, 3(1):66–106, Mar. 1996.

21. M. Roseman and S. Greenberg. TeamRooms: Network Places for Collaboration. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 325–333, New York, Nov.16–20 1996. ACM Press.

22. C. Schuckmann, L. Kirchner, J. Schümmer, and J. M. Haake. Designing Object-Oriented Synchronous Groupware With COAST. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 30–38, New York, Nov.16–20 1996. ACM Press.

23. G. Smith. Cooperative Virtual Environments: Lessons from 2D Multi User Interfaces. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 390–398, New York, Nov.16–20 1996. ACM Press.

24. R. B. Smith. What You See Is What I Think You See. *SIGCUE Outlook*, 21(3):18–23, 1992.

25. M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Transactions on Office Information Systems*, 5(2):147–167, 1987. An earlier version appeared in CSCW '86.

26. ShowMe SharedApp, Mar. 23, 1998. <http://-www.sun.com/products-n-solutions/sw/-ShowMe/products/ShowMe_SharedApp.html>.