Making Application Sharing Easy: Architectural Issues for Collaboration Transparency

Steven L. Rohall
IBM T.J.Watson Research Center
Cambridge, MA, USA
steven_rohall @ us.ibm.com

James "Bo" Begole
Sun Microsystems Laboratories, Europe
Saint Ismier, France
bo.begole @ sun.com

ABSTRACT

For many years, people have wanted to share single-user applications. The vision has been to replicate instances of a single-user application throughout the network and transmit input events from one instance to the others. Although there have been numerous attempts at building such collaboration-transparent sharing systems, many issues remain unresolved. The intent of this workshop is to review the state of the art in application sharing with the goal of identifying how application architecture can better support collaboration transparency.

1. MOTIVATION

1.1 Background

Collaborative use of replicated, single-user applications has long been a dream of CSCW practitioners. If such a system were available, then the myriad of single-user applications could be repurposed as collaborative tools. Not only would people be able to collaborate, they would be able to collaborate with the applications to which they are accustomed.

Today, despite much research in this area, the choices for synchronous sharing of applications are limited. A few applications, such as multi-user games, are developed with integrated collaborative features. The vast majority of end-user applications, however, are written as single-user applications. Users, who may be motivated to add collaborative features to an application, often lack the resources (e.g., the source code) to accomplish this task. As a result, users have had to resort to "screen-scraping" techniques using tools such as IBM Lotus Sametime [11] and Microsoft NetMeeting [8] to share applications. These tools track changes to a computer's screen buffer and transmit the changes as bitmaps to the other collaborators. This is both CPU- and network-intensive, limiting this technique's utility.

A more flexible technique for collaboration is replicated application sharing. In replicated application sharing, separate copies of a single-user application are run on each collaborator's computer. Events from one copy (e.g., keystrokes) are broadcast to the other copies where they are processed as if they had been generated locally, allowing the distributed applications to stay synchronized.

On the face of it, this idea is simple. If all of the collaborators have a copy of the single-user application, then one user can "drive" all the application replicas. Underlying this idea is the notion that, if the same sequence of events (e.g., user input) is sent to replicated instances of the application, then the application state will be manipulated and modified in the same manner in each of the application copies and each collaborator will see the same result. This approach is much more network-efficient than screen scraping systems, since the bandwidth of the input events is small compared to the application output which gets displayed to the user.

There have been numerous attempts to build replicated application sharing systems [4]. MMConf [3], Dialogo [7], and the first version of Rapport [1] were shared windowing systems which captured windowing system input events and transmitted those events to application replicas. However, all of these systems ran into synchronization problems where the replicated applications would be displaying different output.

Crowley, *et al.* talk about four impediments to maintaining state: "differences in initial application state, misordered input events, nondeterministic applications, and latecomers" [3]. Lauwers, *et al.* make a point of noting that "the synchronization problem is tractable when the shared applications are deterministic" [7]. They claim further that an application is deterministic if, starting from the same initial state, the application will generate the same sequence of outputs given the same sequence of inputs. In addition, the application output cannot depend upon the timing between input events. Ahuja, *et al.*, [1] describe the problem with applications that utilize local state and say that "the maintenance of this environmental consistency is not generally possible".

Begole, *et al.*, [2] call these environmental problems *externalities*. More specifically, they define an externality as an input (other than the user) or an output (other than the display) that is external to the application itself. Their Flexible JAMM system handled externalities by exploiting properties of the Java language to dynamically replace single-user application components with specially-written multi-user counterparts, substituting direct calls to the object with remote method invocations on a proxy object. More recent projects (e.g., [9]) have attempted to simplify the problems of handling externalities and hooking application events using techniques such as aspect-oriented programming [6].

1.2 Issues

All of these systems have highlighted a number of difficult issues with application sharing:

- Collaboration-aware applications versus collaboration-transparent sharing. Is collaboration transparency achievable and at what cost? What sorts of applications are amenable to collaboration-transparent sharing?
- Centralized versus replicated architectures. Centralized event dispatching keeps application state synchronized but at the cost of increased latency and poor user experience. On the other hand, replicated systems have

been plagued with state maintenance problems. Is it possible for replicated applications to stay synchronized?

- "Hooking" existing applications. What is the "right" level at which to hook applications: screen buffer, windowing events, or application events? What are the various techniques for hooking applications? Is it really possible to hook high-level application events without intimate knowledge of the application?
- Latency. What is the effect of latency, introduced by both network and processing delays, on the user experience? Can a remote user ever have the same experience as the user driving the application or should techniques for informing the user about latency be adopted? Is it possible to share real-time applications, such as games, in a collaboration-transparent manner? How can performance be maximized?
- State maintenance. How are potential conflicts addressed or resolved (e.g., turn-taking, optimistic evaluation with undo, operational transformation)?
- Application heterogeneity. Does application sharing always imply that the same applications are employed at all sites?

This list represents just some of the issues associated with application sharing. One of the goals of the workshop will be to address these issues as well as to enumerate other issues which are keeping application sharing from being "easy."

2. GOALS

The primary goal of this workshop is to address the question of making application sharing easy. Much of the prior research in this area has been to address shortcomings in the applications being shared (e.g., they do not expose their events, they mingle application model state with view state). In short, sharing these applications is hard because they were written with single-user use in mind. How would they be different if their architects had anticipated application sharing? Since it is unlikely that all future applications will be built to be collaborative, what guidance can we give future application developers for making their applications easier to share? To adequately address this question, we will:

- Discuss architectural issues associated with collaboration-transparent application sharing (as detailed above).
- Describe current solutions to these issues as embodied in research systems.
- Determine what an application architecture that more readily supports collaboration transparency would look like.

Another implicit goal of the workshop is to foster this community and look for ways to coordinate future research and development efforts.

3. ACTIVITIES

The workshop will be run over a full day, with the majority of time spent on discussion and brainstorming. The day will be structured as follows:

- Introductions and presentations. The participants will introduce themselves, list their interests, and present their position on application sharing. Position presentations will be grouped so that similar issues are presented at one time.
- Discussion on architecture. Once a number of issues have been presented and discussed, the workshop will focus on making application sharing easy. What things work well in an application sharing system? How should future applications be architected to better support application sharing?
- Future directions. How should the workshop participants follow up on the day's discussions?

4. ORGANIZATION

4.1 Participation

We seek to invite a maximum of 15-20 participants on the basis of position papers submitted prior to the workshop.

4.2 Submissions

Interested participants will need to submit position papers before September 1st. Each position paper should be no more than 4 pages in standard ACM CSCW formatting. Position papers must include the following sections:

1. Title, names, affiliations, and email addresses of the authors.
2. Description of recent or current work in collaboration-transparent application sharing. The section should describe the issue or problem as well as the architectural solution to the issue.
3. Suggestion(s) for architectural changes in applications which would make sharing easier
4. Short biography of the authors' backgrounds, areas of expertise, and motivation for participating in the workshop

Submissions must be in PDF format, and emailed to steven_rohall @ us.ibm.com. Submissions must include the name, contact, and full address of the author.

Copies of the accepted position papers will be distributed to all participants prior to the workshop.

4.3 Selection Process

The organizers will review all submissions and select participants. Depending upon the number of submissions, participation in the workshop might be limited for submissions with multiple authors. To the extent possible, submissions will be selected so as to present a range of issues and solutions. All participants are required to register to attend CSCW 2004.

4.4 Timeline

- After June 14: Call for position papers.
- September 1: Deadline for position papers

- September 1-29: Review position papers
- September 30: Notification of acceptance
- October: Preparation for workshop

The workshop preparations include preparing the informal proceedings with all of the papers, emailing out the agenda and structure of workshop presentations and discussions, and any additional logistics. If possible, a wiki will be created to allow download of position papers and discussion both before and after the workshop.

4.5 A/V REQUIREMENTS

The organizers will bring own LCD projector; screen to be provided by conference.

4.6 ORGANIZERS

Steven L. Rohall

Collaborative User Experience Group, IBM T.J.Watson Research Center, Cambridge, MA, steven_rohall @ us.ibm.com

Steven Rohall is a software architect at IBM Research. He is currently working on the Zipper [9] system for replicated application sharing. Prior experience in synchronous groupware includes the VIEP [10] system while at TASC and the Rendezvous [5] system while at Bellcore (now Telcordia Technologies). Other research interests include information visualization and electronic mail.

James "Bo" Begole

Sun Microsystems Laboratories, Europe, Saint Ismier, France, bo.begole @ sun.com

Bo is a staff scientist in Sun's Network Communities group, where he focuses on distributed collaboration. Prior to joining Sun, Bo developed the Flexible JAMM [2] system for supporting collaboration transparency.

5. REFERENCES

[1] Ahuja, S.R., J.R. Ensor, and D.N. Horn, "The Rapport Multimedia Conferencing System," *ACM SIGOIS Bulletin*, 9(2-3) April/July 1988, pp. 1-8.

[2] Begole, J., R.B. Smith, C.A. Struble, and C.A. Shaffer, "Resource Sharing for Replicated Synchronous Groupware," *IEEE/ACM Transactions on Networking*, 9(6) December 2001, pp. 833-843.

[3] Crowley, T., P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson, "MMConf: an infrastructure for building shared multimedia applications," *Proceedings of CSCW'90*, Los Angeles, CA, pp. 329-342.

[4] Greenberg, S., "Sharing views and interactions with single-user applications," *ACM SIGOIS Bulletin*, 11(2-3) April/July 1990, pp. 227-237.

[5] Hill, R.D., T. Brinck, J.F. Patterson, S.L. Rohall, and W. Wilner, "The Rendezvous Language and Architecture for Constructing Multi-User Applications," *ACM Transactions on Computer-Human Interaction*, 1(2), June 1994, pp. 81-125.

[6] Kiczales, G., *et al*., "Aspect-Oriented Programming," *Proceedings of the European Conference on Object-Oriented Programming (ECOOP 97)*, Jyväskylä, Finland, pp. 220-242.

[7] Lauwers, J.C., T.A. Joseph, K.A. Lantz, A.L. Romanow, "Replicated architectures for shared window systems: a critique," *ACM SIGOIS Bulletin*, 11(2-3) April/July 1990, pp. 249-260.

[8] NetMeeting: http://www.microsoft.com/netmeeting.

[9] Rohall, S.L. and J.F. Patterson, "Another Look at Replicated Application Sharing," Note submitted to CSCW 2004.

[10] Rohall, S.L. and E.P. Lahtinen, "The VIEP System: Interacting with Collaborative Multimedia," *Proceedings of UIST'96*, Seattle, WA, pp. 59-66.

[11] Sametime: http://www.lotus.com/sametime.