Session Management for Collaborative Applications

W. Keith Edwards

Graphics, Visualization & Usability Center
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
+1 (404) 894-6266
keith.edwards@gvu.gatech.edu

ABSTRACT
Session management systems for collaborative applications have required a great deal of reimplementation work by developers because they have been typically created on a case-by-case basis. Further, artifacts of this development process have limited the flexibility of session management systems and their ability to cooperate across applications, resulting in the fairly formalized, heavy-weight session management found in most collaborative systems today. We present a model for a light-weight form of session management, the theoretical foundation for this model (based on the sharing of information about user and system activity), and details of a collaboration support environment which implements our session management model.

KEYWORDS
Computer-supported cooperative work, collaboration support environments, session management, Intermezzo.

INTRODUCTION

The primary characteristic of collaborative applications is that they, by definition, involve the interaction of multiple users. The manner in which the users of an application join together into a collaborative session is determined by the session management system used by the collaborative application.

The term *session management* refers to the process of starting, stopping, joining, leaving, and browsing collaborative situations across the network [12]. Examples of collaborative situations include a group of users editing a text document in a shared editor, or a video conference among a number of participants.

Currently, developers typically implement subsystems to perform session management on a per-application basis. This trend has resulted in three characteristics of current session management systems which are problematic:

- Much work as gone toward "reinventing the session management wheel" in collaborative applications. Application developers typically implement subsystems to perform session management when they build an application. There is little cooperation between applications or code reuse between developers.
- Since session management itself is subordinate to the centerpiece task the application facilitates, the session management mechanisms in a particular collaborative application are often not very robust, flexible, or powerful. Usually the session management facilities provide the minimal level of functionality to allow the application to perform in a collaborative setting.
- Per-application reimplementation of session management typically means that there are no facilities for altering session management behavior on a global (across-application) scale. For example, it may be difficult to turn off incoming requests for collaboration across *all* collaborative applications running on a user's desktop.

Early in the development of a new class of software, implementation mechanisms tend to be *application-centered*. That is, solutions are implemented at a very low level in the applications themselves. As the problem field matures, and more experience is acquired about the needs of applications, solutions tend to be more *environment-centered*. That is, common APIs and run-time facilities are developed which are shared by all applications. We have seen the trend toward more environment-centered approaches recently in the literature of software support for collaboration[2][4][7][12][13].

In this light, we can see that the characteristics of current session management systems described above are a result of the application-centered approaches taken to developing these systems. All of these problems are similar to those encountered by early applications before the advent of common programming interfaces for developing, for example, graphical applications. Common APIs (whether for GUI development, systems programming, or some other domain) enable applications built on those APIs to be developed more quickly and with greater standardization than would be possible otherwise. Further, APIs which "talk" to a shared runtime facility (such as a window server or an OS kernel) allow coordinated control over the applications built on those APIs; policy can be implemented at the shared entity to affect all of the clients of that entity.

Artifacts of the current application-centered practice have limited the flexibility of session management systems in some non-obvious ways, however. Since application-centered approaches, by definition, tend to limit cooperation between applications (since there are no common facilities to build upon), it is very difficult to build session management systems which permit the easy flow of information between applications using these approaches. Put more directly, the types of session management systems that are easy to build in an application-centered approach may not always be the "best" from the perspective of the user. Building collaborative systems around non-cooperating session management facilities restricts the options available to us as developers of collaborative applications: the tools influence the resulting system. As the state of the practice has influenced session management systems, session management systems have in turn affected the characteristics of applications built using these systems.

This application-centered development model has forced a mode of operation on collaborative applications. Since the session management services typically haven't provided a great deal of coordination between applications, users have been tasked with performing much of the coordination required to begin a collaboration themselves. We shall look more closely at the current models of session management used by most applications, and at new models which can be created when more environment-centered development approaches are used.

There are two goals of this paper. First, we introduce a new model for session management, called *implicit session management*, which provides a more light-weight approach to session management than has been found in typical systems to date. We believe that this approach is useful in a number of situations. We also present a theoretical foundation of the requirements of this type of session management.

Second, we introduce a system which implements our new model of session management and provides general session management services to applications. Our implementation of this model, called *Intermezzo*, is built using a more environment-centered approach than has typically been found in existing systems. In the process of building a session management service which is flexible enough to implement our model of session management, we will implicitly address the three problem areas of session management services which have resulted from the application-centered development strategies used to date.

MODELS OF SESSION MANAGEMENT

To motivate our model of session management, we must first explore how session management systems typically operate in current applications, and introduce some terminology.

Most collaborative applications built to date have taken a "heavy weight" approach to session management. Two of these heavy weight approaches are common:

- **Initiator-based.** Through some sequence of dialogs the initiating user invites other users to the collaborative session. The number of invitations issued can be potentially large, depending on the application and the context of the task. Invited users can accept or reject the invitation. Examples of such a system include MMConf [2] and RTCAL [8]. Initiator-based session management systems fulfill two goals: to notify users of the existence of the collaboration, and to provide a means for rendezvous with the others in the session.
- **Joiner-based.** The initiating user creates a new session; users must find the session by browsing the list of currently active sessions (or know *a priori* that the session will be taking place). Once they know the session handle they can attempt to join the session. Examples of systems which follow this model include Collage [10]. Joiner-based systems typically only provide rendezvous facilities. The participants use some other means to notify each other that a collaboration is in progress.

Both of these approaches are somewhat awkward because they force *someone* (either the initiator or the participants) to do a significant amount of overhead work. In the case of initiator-based session management, the initiator must explicitly invite all participants to the session. In the second joiner-based case, the joiners are required to find the session handle for themselves. If they do not remember that they are supposed to be taking part in a collaboration, it again falls to the initiator to invite the participants to the session (typically via a "low-tech" solution: a telephone call).

We can call these two approaches instances of *explicit session management*. We use the term explicit because the participants in the collaboration are required to take some action (perhaps time consuming) to join the session. This joining action is often only peripherally-related to the task on which the participants are trying to collaborate. For example, for a group of users who wish to work together on a text document, editing the document is the primary focus of the activity. The peripheral task of either inviting group members, or of searching for a "conference" is an artifact of the implementation of the session management facilities offered by the collaborative applications being used.

Explicit session management seems to be useful in situations where there is a high degree of formality or where there is a natural name for the activity. An analogous "real-world" situation might be, "Faculty meeting Wednesday at 2:00PM in room 302." The task is widely known to the participants, is at a well-known location, and embodies a degree of formality.

It is important to realize that explicit session management (and indeed, any form of session management) can affect the way people collaborate. Because starting a collaboration is expensive, informal collaborations are less likely to take place when using an explicit system. Perhaps because of the formal nature of explicit session management systems, collaborative applications based on this technique tend to resemble meetings.

Unfortunately, more spontaneous collaboration is not likely to fit well into this model. Rather than the faculty meeting form of collaboration, what about serendipitous meetings in a hallway or in a breakroom? Some systems have been built to support this form of "chance meeting," although they have typically been communication-oriented "mediaspace" applications [1][5], rather than, say, shared editors. Mediaspace applications usually bypass the entire issue of session management by leaving users connected to each other all the time. This approach seems appropriate for communication-oriented systems, but may be less useful for other types of applications (we do not wish to require users to leave one of every potential collaborative application on their screens in the chance that someone may wish to collaborate with them using that application). We would like to build a framework for session management that works for non-communication-oriented applications but is still lightweight, supports serendipitous meetings, and is as transparent as possible to the user.

Such a system would not necessarily supplant the explicit forms of session management, but would instead serve as a complementing mode of operation which might be useful for a variety of tasks which explicit session management may hinder.

A Light-Weight Model of Session Management

There are a number of situations in which a more light-weight, *implicit* form of session management which requires less initial overhead may be useful. We can see that in many situations invitations are not needed. For example, explicit computer-based invitations are often not needed if the collaboration is within a small group of people who know that they are supposed to be working on some collective task. Often, social pressures will serve to keep uninvited participants out of the collaboration, without the need for strict machine-enforced invitation protocols. And of course, serendipitous collaboration doesn't require invitations by its nature.

But what about rendezvous? What can we do to make the process of joining the collaboration as light-weight as possible? In the models we examined earlier, rendezvous was accomplished by some mechanism orthogonal to the central task at hand (requiring the user to browse a list of conferences before being able to enter a shared editor, for example). A more direct approach would be to have the act of opening the object of the collaboration provide the potential for collaborative activity itself.

Table 1: Session Management vs. Types of Collaboration

Thus, to collaboratively edit a file, users would simply edit the same file. The system would detect the potential for collaboration inherent in the fact that multiple users are working on the same object, without the need for naming sessions or browsing lists of sessions to accomplish rendezvous. We call this process *implicit session management* because it avoids the overhead of the explicit session creation, naming, and browsing phase. In contrast to the explicit forms of session management (initiator-based and joiner-based), this form of implicit session management is *artifact-based*.

There are physical analogs for this type of collaboration:

- In "old fashioned" libraries with paper card catalogs, a person would know if another individual was interested in similar subject matter by the proximity to them when browsing the card catalog. A potential (although not required) collaboration exists ("I see you're interested in sailing as well."). (Thanks to David Gedye for this example.)
- In a medical office, one worker may know that another is updating a patient file because the file is not in its place in the cabinet (in many offices a paper slip is left in its place denoting who has the file in question). If desired, the worker searching for the file may go to the one who has it to share information.

In both of these cases, participants know that they're working on the same or similar tasks because they are interacting with the same physical object. Rendezvous is based on the sharing of this common object. Collaboration is trivially easy, although not required. Table 1 lists some characteristics of collaboration which may make explicit or implicit session management more effective.

We have described how implicit session management would look from the user's perspective, but have not yet addressed the issue of what system-level facilities are required to implement such a model. The next section presents a theoretical model for a service which can automatically provide implicit (as well as explicit) session management facilities to applications.

ACTIVITY INFORMATION AS A FOUNDATION FOR SESSION MANAGEMENT

It is our thesis that *activity information* can serve as a foundation for building a powerful and flexible session management service, with application beyond that of implicit session management only. Activity information is information which contains details of the current tasks which are being run across the network: the users on the systems, the applications or tasks they are currently engaged in, and the objects of those tasks (that is, the data on which the applications are operating).

At any given point in time there are a number of activities which exist across the network. We can think of an activity as a tuple:

$$A_n = (U_n, T_n, O_n)$$

Where A_n represents the *n*th activity. This activity is comprised of U_n, T_n, and O_n which respectively represent the *n*th user, task (application), and object (data).

For the purposes of session management, U_n and T_n can be thought of as simply tokens which uniquely identify the user and the application he or she is working with. The mappings between users and applications, and their tokens U*n* and T*n*, must be one-to-one.

Objects are somewhat more complicated, since different applications may operate on different data domains. For example, an editor may operate on a text file, while a calendaring program may operate on a section of an appointment database. Each of these data sets possess fundamentally different semantics of use and representation.

Thus the O_n token consists of a *namespace identifier* and a *name* which is valid within that namespace. The namespace defines the type of the data set the application is operating on. Examples of namespaces include files, database selections, and so forth. Each namespace defines a set of names which have a one-to-one correspondence with the objects they represent. For example, a particular file is represented by a unique name within the namespace *file*.

Implicit Session Management

Applications that wish to participate in implicit session management must publish activity information so that it can be made visible to the session management service. When performing implicit session management, the session management service will automatically detect potential collaborative situations and take "appropriate" action (as described to it by applications or user preferences). There is no need for users to explicitly issue invitations or create sessions.

The session management service detects potential collaborative situations by looking for overlaps or confluences in the activity information published by applications across the network. When two activity tuples exist which contain the same object token (that is, when both the namespace and the name of two objects match exactly), then the session management service can take action to allow the users to enter into a collaborative situation.

For example, if two users edit the same file, the session management service can notify the users of this fact and allow them to easily enter into a "spur of the moment" collaboration. The mechanics of joining a collaborative endeavor closely match the human dynamics of collaboration. When two coworkers wish to work on a paper together, one will typically say, "Let's get together sometime after lunch and finish up the budget." No *formal* invitations are issued, and no name is given to the activity. Instead, the coworkers simply begin working on the budget at or about the same time. The action of working on the same budget implicitly carries with it the notion of collaboration. Whereas in the explicit forms of session management the burden of labor is on the users of the system, in implicit session management the system itself can assume the task of detecting and handling potential collaboration.

Note that this form of implicit session management, because it is so transparent, requires applications or the collaboration support environment to provide powerful mechanisms for policy controls to allow users to enable, disable, or otherwise alter the behavior of the session management service. Obviously we don't want to automatically be thrown into a shared editor anytime we happen to open a file that another person has open.

The mechanisms for publishing and retrieving activity information, for generating unique object names within a namespace, and for taking appropriate action upon detection of potential collaboration are defined by particular implementations of this model. We now describe our system which implements implicit, artifact-based session management using the model of activity information described above.

INTERMEZZO IMPLEMENTATION

Intermezzo is a collaboration support environment [3] which is addressing issues related to session management and other collaborative activities. The goal of the system is to facilitate the sharing of "coordinating" information to link multiple collaborative applications into a more holistic environment. Session management is one form of coordinating information which Intermezzo makes available to applications.

Intermezzo provides a set of run-time services, programming libraries, and conventions that applications can use to participate in a session management service, among other things. Intermezzo uses a publish/subscribe model to make information available across the network. A simple object database manages the information published by applications and makes it available to interested parties [6].

There are three main abstractions used in the Intermezzo implementation: `Threads` (abstraction for computation), `Ports` (abstraction for communication), and `Resources` (abstraction for data). These abstractions are used throughout Intermezzo: both in the run-time component, and in the programming interfaces used by application developers.

Communication between threads is handled by `Ports`. A Port is an abstraction for a half-duplex communication channel. Port subclasses can implement particular transports and semantics. Some types of Ports can only transfer data between threads in a single address space (such as `MessageQueue` ports, which connect the threads which make up the Intermezzo run-time service), while others can be used for communication between threads in different address spaces (such as `RemoteProcedureCall` ports, which are used to connect client applications to the Intermezzo server).

The data objects that Intermezzo manages are called `Resources`. Resources are essentially objects which contain lists of attributes (key-value pairs) and have types and unique IDs. `Resources` maintain notions of ownership (the application which created a given resource) and permissions (who is allowed to view and update given resources). All resources can "pack" and "unpack" themselves in a number of data interchange formats for transmission over the network into another address space

Intermezzo Programming Interface

Intermezzo provides a client-side library which application developers can use to interact with the Intermezzo run-time service. This library uses the same abstractions used internally by the run-time service (`Threads`, `Ports`, and `Resources`). A `Client` thread is automatically instantiated by the library to handle communication to the server process. The components of the activity information model presented above map directly into resources in Intermezzo: users, tasks, and objects are all resources. A special "container resource" called an `ActivityRecord` holds references to `User`, `Task`, and `Object` resources which together represent one activity.

`Object` resources maintain their namespace and name as attributes. The client-side code in Intermezzo is responsible for generating unique, network-wide names for any given object within a particular namespace.

Whenever an "Intermezzo-aware" application is started, it publishes an `ActivityRecord` resource representing itself, its user, and the data it is operating on. Essentially, enabling this behavior requires the insertion of one line of code into the application's start-up routine. This is all that is required to generate activity information which may be used by other applications, and by the Intermezzo session management facilities.

Applications which wish to be "better behaved" under Intermezzo should take several more actions, however:

- Request and handle notification from the server whenever a potential collaboration exists.

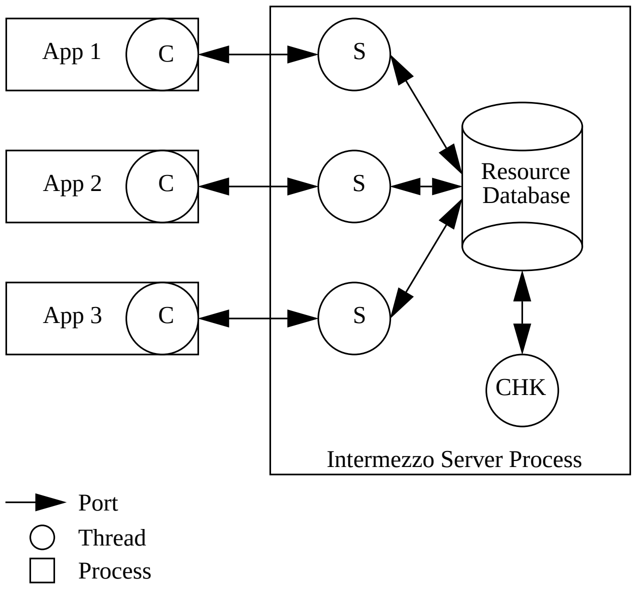

Figure 1: The Intermezzo Run-time Architecture

- Notify the server about any policy constraints desired by the user or the application itself.

Further, applications are free to use the resources they have published for other purposes, such as storing user preferences or application defaults. Applications can update and access the information stored in these resources as long as they have the appropriate permissions.

Intermezzo Run-time Architecture

The Intermezzo run-time system is built as a multi-threaded server (see Figure 1). A `Server` thread is instantiated once for every connection to a client, and handles servicing requests and replies to and from the client application. Other threads in Intermezzo include a `Main` thread (responsible for creating other threads as needed), and a `Checkpointer` thread (which can periodically flush the database to stable storage).

The Intermezzo run-time service stores resources in a simple object database. Whenever the database is updated, Intermezzo searches for `ActivityRecords` containing references to objects which have the same namespace and name. This search process is implemented via *triggers*, a technique developed in *access-oriented* programming [14] that allows modifications of a data value to automatically cause some action to take place.

The particular actions that Intermezzo takes when it detects a potential for collaboration depend on the "intelligence" of the application, and the desires and preferences of the users.

In the case of collaboration-aware applications, Intermezzo may be configured to generate events to the applications (say, the text editors being used to work on a budget). The applications receive these events and go into “shared” mode.

In the case of collaboration-naive applications, Intermezzo can generate a sequence of messages to a shared window system to instruct it to begin sharing the applications, even though they may have been written as single-user tools.

The Intermezzo server is capable of dynamically loading new thread objects into its address space at run-time. This feature makes the server extensible to new behaviors without the need to recompile the entire system.

Policy in Intermezzo

Of course, applications can implement their own policies on top of the information provided to them by Intermezzo. Even though an application may receive an event from Intermezzo indicating a potential collaboration exists, the application is not required to act on it. The application may go directly into collaboration mode, ask the user if it should go into collaboration mode, or ignore the event entirely.

Note that it is possible to use the Intermezzo facilities for implementing more traditional, explicit session management. Applications simply tell Intermezzo to not generate collaboration events, or ignore them when they are received. The process of browsing users and activity across the network simply becomes a select operation on the database to search for the data the application is interested in.

Intermezzo adds power to files and other system objects in much the same way that graphical user interfaces have: clicking on a document in a GUI “magically” launches an application and loads that file. What had, prior to the advent of GUIs, been a relatively static, inanimate object suddenly acquired a new property: touching it causes some action to take place in the system. Intermezzo adds another property to files: touching them can potentially place the user into a collaborative situation with the other users who are accessing that file.

By providing mechanisms which allow applications to access session management services, Intermezzo addresses several of the problematic characteristics of typical session management services:

- By providing a library to application developers to implement session management, Intermezzo keeps developers from having to rebuild session management facilities on a case-by-case basis.
- The Intermezzo session management facilities are flexible enough to support a range of applications, and typically provide greater power than the session management subsystems provided by current applications.
- As a central point of control, Intermezzo allows users to easily change session management behavior across applications.

An Example: File Objects

An example may help clarify how object resources are generated and used for session management. The most common type of object which is used by applications is the simple file.

When an application which operates on files (say, a shared editor) is started, Intermezzo publishes an `ActivityRecord` which represents the user of the application, the application itself, and the file the application is operating on.

The `Object` resource is constructed with a namespace of `File`. A name is generated which can be used to uniquely identify this file across the network.

In our implementation (which is on Unix), the unique name for a given file *f* is constructed from the hostname *H* on which the filesystem containing the file is mounted, the name of the filesystem *F* the file resides on, and the inode number *I* of the file on that filesystem.

$$ame_f = (H_f, F_f, I_f$$

This scheme (which is similar to that used by [11]) allows us to create an identifier for a file which has the following properties:

- The Intermezzo names for two different files are guaranteed to be different, even if the pathnames used to refer to the files are the same.
- It is impossible to generate two distinct names for the same file, even if different pathnames are used to refer to the file (or if the file is remote mounted on another machine).

That is, there is a network-wide unique one-to-one mapping between Intermezzo file names and actual files.

Once the `User`, `Task`, and `Object` resources have been created, they are published as part of a new `ActivityRecord` resource which represents one instance of a particular activity (a single user working with an application on a particular data object). The Intermezzo run-time service receives the new `ActivityRecord` and searches its database of resources, looking for a confluence in the `Object Name` and `Namespace` attributes.

Intermezzo assumes that applications and users will participate in a number of conventions that determine the activity which will be taken when a confluence is found. Whenever a confluence occurs, Intermezzo will retrieve an attribute called `Colab Action` on the `Task` resources associated with the two overlapping `ActivityRecords`. The `Colab Action` attribute is used to tell Intermezzo how to handle the potential collaboration. Possible courses of action include generating events to the application, or running an arbitrary program. Application developers can decide how their applications will handle potential collaborations by using different values for the `Colab Action` attribute.

Intermezzo also examines the value of the `Colab Allow` attribute on the `User` resources associated with the two overlapping `ActivityRecords`. By convention, Intermezzo treats the `Colab Allow` attribute as a list of users with whom collaboration will be initiated whenever a potential for collaboration exists. If the "calling" user is not in the `Colab Allow` list for the "receiving" user, then Intermezzo takes no action upon detecting the potential collaboration. Users can exert control over the process of collaboration by changing the value of `Colab Allow` on the `User` resource which represents them.

USER PERSPECTIVES

Our implementation provides the desired features at the user level. By handling the coordinating of user activity information internally to the session management service, we do not force the users of collaborative applications to perform the coordinating tasks themselves. Further, the system is very flexible in the actions it can take when a potential collaboration occurs.

Intermezzo satisfies the goals of supporting light-weight, transparent collaborative rendezvous in which the act of accessing an object provides the trigger for collaboration. There is no need for an orthogonal set of session management actions which the users must use to enter into a collaborative task.

Further, Intermezzo supports serendipitous collaboration in non-communication-oriented applications. Awareness is enhanced through the use of serendipitous encounters.

Finally, the facilities available to Intermezzo aren't limited to supporting implicit session management only. Applications which need heavy-weight, explicit forms of session management can also be constructed using Intermezzo. The system provides a generally useful software substrate which is sufficient for implementing an array of policies and mechanisms.

OTHER APPLICATIONS

There are a number of applications of the models we have developed for Intermezzo.

Activity Information as Input to Users

We can use the collected information about activity to provide awareness about users across the network. Users can know if a coworker is working on an important document and should not be interrupted. A number of activity monitor applications have been built which provide this service. (for example, [9]). Used in this way, activity information is an input to users which allows them to make judgments about whether or not to interact with a colleague.

Activity Information as Input to Applications

One aspect of activity information which has not been fully addressed is the use of activity information as input to applications. If activity information is widely available, applications can be written to take user activity into account. For example, if a user is already engaged in a video conference, an application may decide to use pop-up notifications to the user, rather than audio notifications which may potentially disrupt the conference.

Per-User and Per-Application Data Storage

Since Intermezzo creates objects to represent users and applications, those objects are obvious places to store information about user preferences, "personal data," and application defaults. Intermezzo provides an API which can be used to access this information across a network.

Data Interchange

The facilities provided by Intermezzo can be used as a general-purpose data interchange facility. While not suitable for high-bandwidth transmission, applications which need to exchange relatively small amounts of data, or have data storage needs which are closely matched to the services provided by an object database may find Intermezzo useful.

STATUS

Both the Intermezzo programming interface and the run-time service are implemented in C++. The total system is approximately 25,000 lines of code. We are currently using Transport-Independent Remote Procedure Calls for communication between clients and the server. Our implementation is based on the Solaris 2.3 operating environment, and runs on Sun SPARCstations.

SUMMARY AND FUTURE DIRECTIONS

We have presented a model in which activity information may be used as a basis for session management in collaborative applications. The use of activity information allows us to achieve two goals: we can support very light-weight implicit forms of session management, and we can increase the overall flexibility of all forms of session management by using activity information as an input to both users and applications.

Our system, Intermezzo, provides a software substrate for information storage and retrieval in a network setting, along with programming interfaces and conventions for information sharing in collaborative applications. Intermezzo has proven to be a powerful tool for implementing the session management model described here.

Currently our work on Intermezzo is focusing on a number of directions. First, more powerful mechanisms for policy control are needed. We are investigating formalisms to express policy constraints in collaborative settings. Second, we are investigating implementation issues related to Intermezzo. Our publish/subscribe object database approach has so far proven sufficient. We are investigating how scalable our approaches are, and what types of implementations might be needed to support further types of information sharing.

Usability studies to determine the impact and applicability of different forms of session management will be required to assess how these models of session control can be used effectively in collaborative systems.

We plan on making a public release of Intermezzo once the code is fully stable and portable, and once we have experience building a number of testbed applications on the system.

ACKNOWLEDGEMENTS
This work is sponsored by Sun Microsystems, Inc. We are grateful for their generous support.

REFERENCES

[1] Ahuja, S.R., Ensor, J.R., Horn, D.N. "The Rapport Multimedia Conferencing System." *Proceedings of Conference on Office Information Systems*, Palo Alto, CA: IEEE, 1988, 1-7.

[2] Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. "MMConf: An Infrastructure for Building Shared Multimedia Applications." *CSCW 90: Proceedings of the Conference on Computer-Supported Cooperative Work*, Los Angeles, CA: ACM, 1990, 329-342.

[3] Dewan, P., and Choudhary, R. "Flexible User Interface Coupling in a Collaborative System." *Reaching Through Technology, Proceedings of ACM CHI '91: Conference on Human Factors in Computing Systems*, New Orleans, LA: ACM, 1991, 41-48.

[4] Dewan, P., and Choudhary, R. "Primitives for Programming Multi-User Interfaces." *UIST 91: Proceedings of the ACM Symposium on User Interface Software and Technology,* Hilton Head, SC: ACM, 1991, 69-78.

[5] Dourish, P., and Bly, S. "Portholes: Supporting Awareness in a Distributed Work Group." *Striking a Balance, Proceedings of ACM CHI '92: Conference on Human Factors in Computing Systems*, Monterey, CA: ACM, 1992, 541-458.

[6] Edwards, W. Keith. *Intermezzo Implementation Notes*. Georgia Tech GVU Center Technical Report GIT-GVU-93-42, 1993.

[7] Gibbs, S.J. "LIZA: An Extensible Groupware Toolkit." *Wings for the Mind, Proceedings of ACM CHI '89: conference on Human Factors in Computing Systems*, Austin, TX: ACM, 1989, 29-36.

[8] Grief, I., and Sarin, S. "Data Sharing in Group Work," *Computer-Supported Cooperative Work: A Book of Readings*, Irene Grief, ed. San Mateo, CA: Morgan Kaufmann, 1988, 477-508.

[9] Manandhar, S. "Activity Server: You Can Run But You Can't Hide." *Multimedia for Now and the Future: Proceedings of the 1991 USENIX Conference*, Nashville, TN: USENIX Association, 1991, 299-312.

[10] *NCSA Collage for the X Window System User's Guide*, National Center for Supercomputing Applications.

[11] Patel, Dorab, and Kalter, Scott D. "A UNIX Toolkit for Distributed Synchronous Collaborative Applications." *Computing Systems*, Berkeley, CA: University of California Press, 1993, p. 105-134.

[12] Patterson, J. F., Hill, R. D., Rohall, S. L., and Meeks, W. S. "Rendezvous: An Architecture for Synchronous Multi-user Applications." *CSCW 90: Proceedings of the Conference on Computer-Supported Cooperative Work*, Los Angeles, CA: ACM, 1990, 317-328.

[13] Roseman, M., and Greenberg, S. "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications," *Sharing Perspectives: Proceedings of the Conference on Computer-Supported Cooperative Work*, CSCW'92, Toronto, Ontario: ACM, 43-50.

[14] Stefik, M.J., Bobrow, D.G., and Kahn, K.M. "Integrating Access-Oriented Programming into a Multiparadigm Environment." *IEEE Software*, 3,1, IEEE Press, January, 1986, 10-18.