Providing Flexible Services for Managing Shared State in Collaborative Systems

Hyong Sop Shim, Robert W. Hall, Atul Prakash, and Farnam Jahanian
Department of Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor, MI 48109-2122 USA
E-mail: {hyongsop,rhall,aprakash,farnam}@eecs.umich.edu

To effectively collaborate in Internet environments, it is critical to efficiently manage the shared state of collaboration. However, the management of shared state is highly situational; different collaboration semantics require different measures tailored to their specific needs. Hence, providing a general set of services that meet the management requirements of varying collaboration situations is challenging. In this paper, we discuss our approach to providing such services. The services are made flexible by allowing collaborators to choose appropriate services based on the needs of their collaboration tools and specific characteristics of their shared state. We present the shared state management services provided by our Corona server that embodies our approach and report experience with its use.

Introduction

Computer-supported collaboration often requires sharing of certain application context by geographically dispersed participants. The collaborators are able to *work together* over distance by making changes to a shared state and observing the changes made by others.

The semantics of shared state is *application-dependent*. For example, in a group-drawing tool, the shared state may be defined as the contents of the canvas. In a window-sharing environment, such as the one supported by DistView [21], the shared state includes the attributes of a shared window, e.g., the size of the window, and the internal states of application-specific objects associated with the window.

On the other hand, CSCW system developers need *application-independent* services for managing shared state and providing awareness information about its use. Critical issues concerning the management of shared state by application-independent services include: **Awareness:** In a computer-supported collaboration, a user may be unaware of the presence of other participants or their current status without explicit support from the underlying system. Collaboration awareness information such as when users join/leave a collaboration session, whether or not they are paying attention when connected (e.g. gone out of the office for coffee), or when they are disconnected from a session due to network or client failures plays an important role in managing shared state as it may dictate the interactions of collaborators. Such awareness information should be available for ready access.

Synchronization: A shared state should be consistently synchronized for the entire duration of a collaboration. Furthermore, collaborators should be allowed to access and modify the shared state concurrently without disrupting each other's work.

Predictable Performance for Late-comers: In synchronous collaboration, participants may join an ongoing collaboration activity. A late-comer should be able to receive a consistent state of collaboration in a "predictable" amount of time - independent of failures of or speed of other clients in the system. Our experience with CSCW systems indicates that users get impatient with a system if it takes them longer than "normal" to join a collaborative session, say, because of failures of other clients or slow bandwidth to a client that might have been selected by the system to transfer shared state. Users expect a "predictable" response time (i.e., limited largely by their bandwidth to the network and the size of the state) for state transfer when they join the system. Conversely, existing users do not want the joining of late-comers to be intrusive or disruptive to their on-going work.

Persistence: A group activity may involve both synchronous and asynchronous collaboration. It is often the case that collaborators may not accomplish all their goals in a single session; they may have to adjourn a session and reconvene at a later time. In such cases, it may be necessary to save part or all of a shared state persistently to be retrieved for a later session. Persistence is beneficial to both synchronous and asynchronous collaboration.

Time-stamping: Users may often want to know *when* an update to a shared state took place. In the chat application of the UARC project, for instance, our users demanded that all messages be time-stamped by a reliable service. Time-stamping can also be useful if users want to know what has changed since the last time they participated in a long-term collaborative session. Hence, mechanisms should be provided to reliably time-stamp updates on the shared state.

Interactive Responsiveness: Users expect collaborative applications to have similar response times as single-user applications. The ability to collaborate should not disrupt the fluidity of users' interactions with their applications.

Client-Based Semantics: The interpretation of the semantics of a shared state should be the responsibility of collaborating application processes. This allows shared states to be scalable to a large number of collaborating processes, and processes of different applications may work over the same shared state [19].

Robust Collaboration: A collaboration session should be robust. It should tolerate various failures of collaborators' host machines and network connections and continue to support the work of non-faulty collaborators. In addressing the above issues, different collaborative applications require different approaches to managing shared state based on their needs. For example, persistence may not be required in all collaborative applications. Also, different users in the same collaborative application may have different awareness needs. Nevertheless, it is highly desirable for the efficient development and widespread use of computer-supported collaboration technology to have a general set of shared state management services that adequately address all the aforementioned issues. Such services should be flexible in that the subscribers to the services are able to select only the services they need, with corresponding overheads.

In this paper, we present our approach to managing shared state. Our approach is realized in a set of shared state management services provided by our Corona server that we have implemented as part of the UARC project's Collaboratory Builder's Environment [14]. The server supports both synchronous and asynchronous collaboration over the World Wide Web, where collaborating clients may be dynamically downloaded over the Internet. In an earlier paper [10], we discussed communication requirements supported by the Corona server and the scalability aspects of communication for different kinds of groups. In this paper, we focus on the management of shared state by the Corona server to address the above issues.

The remainder of the paper is organized as follows. We first discuss the motivation for our work. We then provide a detailed discussion of our approach. This discussion defines basic concepts fundamental to our shared state management services and describes each service in detail. We then report on the implementation status as well as some usage examples of Corona. We conclude the paper by comparing our work with existing systems and by outlining our future plans.

Motivation

Our work on the management of shared state in computer-supported collaboration has its origin in an NSF-sponsored project, called the Upper Atmospheric Research Collaboratory or UARC [8]. The UARC project focuses on the creation of an experimental testbed for wide-area scientific collaboratory work. This testbed is implemented as a large object-oriented distributed system on the Internet and provides a collaboratory environment in which a geographically dispersed community of space scientists perform real-time experiments at a remote facility in Greenland without having to leave their home institutions. This community of space scientists has extensively used the UARC system over the last three years and has expressed a high degree of satisfaction with its mechanisms for remote collaboration. Figure 1 shows a snapshot of various UARC collaboration tools that enable space scientists to remotely conduct their science.

Consider two collaboration tools: the multi-party chat box for exchanging textual message and graphical images and a shared windows facility for viewing instrument data. These two collaboration tools have very different requirements for managing shared state. The chat box allows scientists to exchange textual messages and graphical images within the editing area of the chat box. All the messages are shown in the display area of the chat box. Made possible by our DistView toolkit [21], the shared viewer facility allows the selective sharing of the data display windows of instrument data viewers. In order to share a window, a scientist first *exports* the win-

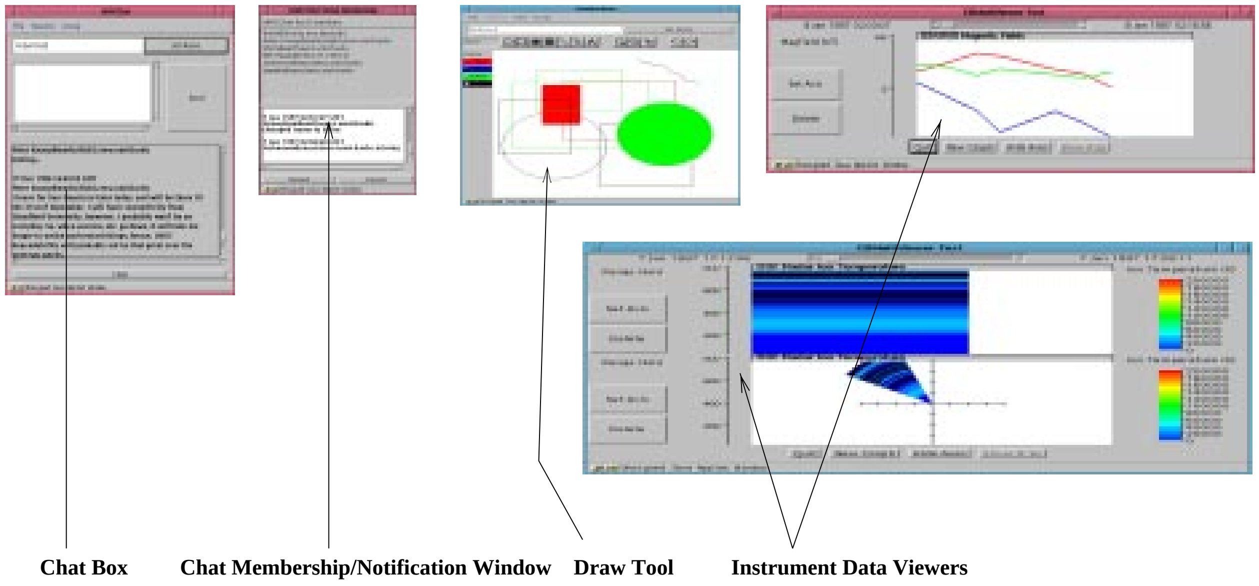

Figure 1. ADD CAPTION

dow to a public repository. The exported window may subsequently be *imported* by any scientist who is interested in collaborating with the exporter of the window. Henceforth, the shared window provides the collaborating scientists with a synchronized view of the data being displayed and ensures that its physical appearance on the screen, e.g. its size, remains synchronized throughout collaboration.

The above two collaboration tools demand different strategies for managing shared state. The shared state of the chat box is defined to be the list of messages exchanged between participants. The shared state of a viewer includes the range of data being displayed in the shared window, the graphical display of the data itself, the settings of the display, and all of the graphical attributes associated with the window. In order to maintain the synchrony of its shared state, the chat box only needs to append each new message to its queue of exchanged messages; it does not have to re-send previously exchanged messages when updating its state with a new message.

In contrast, when a participant resizes a shared window, all instances of the window have to be resized. Likewise, when a scientist changes the display mode for the data being displayed in the window, the other replicas of the window have to reflect the setting change by overwriting their respective internal variables with the new value. Hence, keeping shared windows synchronized requires replacing part of its shared state with new values.

Another distinction concerns the degree of synchronization. The scientists are not very concerned about the total order of messages shown in the display area of the chat box, especially when the messages are timestamped. Instead, they want to interact with each other without delay by being able to exchange messages as freely as possible. When sharing a window, however, concurrent updates on the shared window should be serialized. Otherwise, the shared state of the window seen by different scientists may be inconsistent. For example, the shared window may be of different sizes, or the instrument data displayed in the window may be in different modes for collaborating scientists.

Over the years, the design of the UARC system has evolved through several generations. The current design is an applet-based architecture implemented in Java. It takes advantage of the widespread use of the World Wide Web and the platform-independence of Java applets. A key component of the UARC system is the Corona server. The server embodies our approach to shared state management; it is powerful and flexible enough to meet the varying shared state management needs of UARC applications as well as general collaboration environments.

Corona Shared State Model

Ellis, et al. define the shared state of applications in computer-supported collaboration as "...a set of objects where the objects and the actions performed on the objects are visible to a set of users" [7]. It follows that the management of a shared state may be defined as the management of actions, i.e. accesses and updates, on the objects that constitute the shared state.

We require each object in a shared state to have an identifier to distinguish the object from the others in the same shared state. The object should also be able to write its internal state to a stream as well as set its state upon receipt from a stream. The latter requirement is commonplace in object-oriented applications that utilize persistent objects. The identifiers for objects in a shared state may be automatically generated by a support system and may not incur extra programming efforts. For example, the object instances of the DistViewObject class in our DistView toolkit [21] are automatically assigned such identifiers by the DistView runtime system.

Corona Server Overview

In this section, we describe the basic approach of the Corona server to shared state management. We begin the discussion by defining the concept of *group*.

Group

The basic unit of collaboration in the Corona server is a group. A group is defined to be a set of application processes, termed *members*. A group may be characterized as *stateful* or *stateless*. A stateful group is associated with a shared state; the members of the group collaborate with each other by collectively accessing and modifying the shared state of the group. To participate, an application process has to join the group first. Once a member leaves the group, it no longer receives any update notifications on the shared state. A stateless group has no shared state and is used for group administrative tasks such as dissemination of group membership notifications.

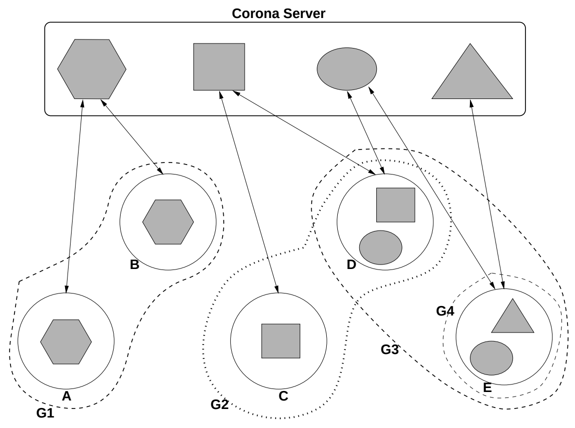

Figure 2. Overview of the Corona Server. Circles represent clients, dotted lines depict groups, and different shapes represent different shared states. Clients may belong to different groups; Client D belongs to both Group G3 and G4 etc. All the groups are stateful.

Figure 2 illustrates the group concept.

Groups may also be characterized as *persistent* and *temporal*. A persistent group exists even when it has no members; if it is stateful, the shared state of the group also persists. A temporal group ceases to exist when it has no members; if it is stateful, its shared state is also lost.

Note that our notion of group is distinct from the concept of a group of users who may be engaged in various collaboration activities. It may be viewed that our group represents a particular collaboration activity that a set of users is presently participating in.

Group State Management

The server manages a set of groups. A group is created by a qualified client sending a request to the server [1]. The client specifies the name of the new group and whether or not the group is stateful.

If a group is stateful, the client may also send the initial state of the group to the server. The server represents the shared state of a stateful group as a table indexed by object identifiers. Each table entry is a *byte stream* representation of the state of each object in the shared state. The server keeps the shared state table up-to-date as the group members update the shared state. When a new member joins the group, the server transfers the contents of the table to the new member. By default, a stateful group is persistent whereas a stateless group is temporal. The server allows groups to dynamically change their persistence status as needed.

Note that the shared state of a stateful group specifies *what* is shared but does not dictate *how* the shared state is actually used. Instead, the interpretation of the

[1]The server works in conjunction with our session manager as presented in [14], and the session manager determines who may have a privilege to create groups.

semantics of the shared state is left to collaborating processes. This is important because we want our shared state management services to be applicable to a wide range of collaboration situations. The lack of knowledge of the semantics of a particular shared state frees us from having to deal with its behaviors in a specific situation. Patterson, et al. describe this property as *client-based semantics* [19].

The Corona server takes a centralized approach to providing administrative services for maintaining shared state. A different approach would be to replicate management responsibilities among collaborating clients [3, 11, 13]. In theory, such a replicated approach offers advantages over a centralized counterpart, especially in the issue of fault tolerance. However, in practice, a replicated approach would not be suitable in a wide-area, heterogeneous collaboration environment such as the World Wide Web. For example, maintaining the synchrony of replicated data among all the keepers of the data would be very expensive in such an unreliable environment. Further, clients may run on hosts of unpredictable processing resources and network connectivity. Hence, it would be unreasonable to expect clients to always perform shared state management tasks reliably.

A rationale for centralizing administrative services for managing shared state is that servers are designed to support multiple users and hence tend to be more reliable, allocated more resources, and run in a more controlled environment than clients [19]. Clients usually support the work of a single user and typically are not trusted to be reliable. Furthermore, a centralized service provider such as the Corona server can be made fault tolerant by having multiple replicas of the provider. There exist a number of well-known replication strategies, including a primary-backup approach (passive replication) [16] and a state machine approach (active replication) [1].

Further advantages of a centralized approach are that it provides a single point of serialization and that it simplifies accommodation of latecomers [3, 19]. Further, the detection and handling of faulty clients is easier with a centralized approach than a replicated counterpart where all the clients would need to run a complex membership protocol to account for faulty clients.

Corona Services

The Corona server provides the following set of services for its groups: *group awareness*, *state transfer*, *group multicast*, *lock management*, and *logging*. Figure 3 graphically represents the Corona services.

Group Awareness Service

The group awareness service of the Corona server maintains information on the membership of groups. When a new client joins a group, the server sends a join notification to the other group members. Likewise, when a client leaves its group, the server notifies the other group members of the client's departure. The changes of member attributes are also notified; for example, when a member changes his or her user-

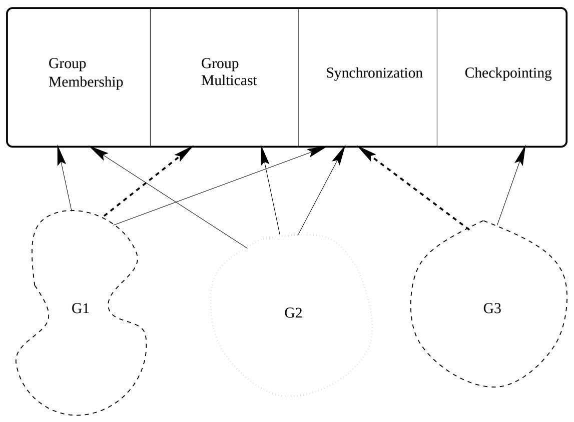

Figure 3. Corona Services. The arrows represent subsriptions to services. Group G1 is subscribing to group awareness/membership, group multicast, and synchronization services. The dashed arrows show that Group G3 is using the synchronization service differently from Group G1 and that Group G2 is utilizing the group multicast service differently from Group G1.

name attribute, the change is broadcast to the other members. Furthermore, a member may indicate to the other members its participation status as *idle* or *active*. An idle member is unable or unavailable to participate in the group activities for some reason, e.g. has to answer a phone call, whereas an active member can fully participate. The server sends participation status notifications for the benefit of the group; the server does not distinguish idle members from active members. By default, a new member is active.

In some collaborative environments, it may be necessary to differentiate users based on the type of operations they can perform on the shared state. It may be desirable to grant only certain users the privilege to update the shared state while others are only allowed to view the changes on the shared state.

The Corona server supports the following member roles: *principal*, *observer*, and *membership-observer*. Principals have update privilege on the group's shared state. Observers and membership-observers are casual members who may only view the updates on the shared state or the changes on the group membership, respectively. Hence, principals need a higher level of awareness of each other's work and presence, and thus require a higher quality of service (QoS) from the server [10]. For example, the server may have to ensure that a principal receives all of its notifications whereas it may notify observers without ensuring that all messages are reliably delivered.

By default, a client is a principal member. A client may also specify its role when it joins a group. Furthermore, the server allows group members to change their roles on the fly in response to collaboration needs or system events.

Group Multicast Service

Clients use the group multicast service of the Corona server to broadcast updates on a shared state; a client sends a message containing update information to the server, and the server broadcasts the message to the members of a group to which the sender belongs. Updates on a shared state may be characterized as either *memoryless* or *memoryful*. An update contains some value that represents a change of state of an object in the shared state. A memoryless update contains a value that *replaces* the old state of the object. In contrast, the value in a memoryful update is *added* to the existing state of the object. Hence, memoryful updates entail incremental changes to the state of the object whereas memoryless updates require overwriting the existing state of the object.

Memoryful updates are useful where the history of changes to the shared state is important to collaboration. In a group-drawing session, for instance, it is undesirable to remove existing shapes whenever a new shape is drawn on the canvas. On the other hand, memoryless updates are useful where the past history of changes is irrelevant to collaboration. For example, to know the current size of a shared window does not mandate knowing the past sizes of the window.

A group multicast message for an update specifies the group name, the object identifier to which the update applies, the byte stream encoding of the update value, and a flag specifying whether the update is memoryful or memoryless. Upon receipt of a memoryless update multicast, the server uses the object identifier to index into the group's state table and replaces the existing stream with the new update stream. Upon receipt of a memoryful update multicast, the server simply appends the update stream to the existing stream. The server then forwards the update message to group members.

The server provides two forms of group multicast: *sender-inclusive* or *sender-exclusive*. Sender-inclusive multicast is used when the server needs to perform additional operations on the message prior to delivering it to the clients, e.g. timestamping the messages of the chat box. With sender-exclusive multicast, the sender applies the update locally and then sends the update to the server which then forwards the update to the remaining group members. We use the phrase multicast instead of broadcast to highlight the fact that update messages may be sent to a subset of the total membership based on the roles of the individual group members.

State Transfer Service

When a client joins a stateful group, the Corona server transfers the current shared state of the group to the new member. A decision has to be made as to what of the shared state should be transferred. Depending on the semantics of collaboration, the new member may be sent only part of the shared state. For example, the shared state of a shared window includes the states of all the user interface as well as application-specific objects associated with the window. Depending on particular collaboration needs, the importer of the window may be given the current states of both user in-

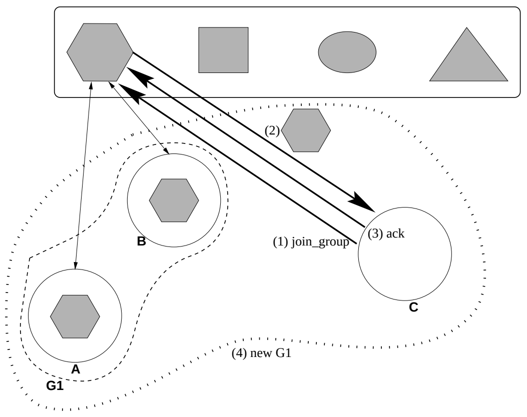

Figure 4. Fast State Transfer in Corona. A new client C is joining Group G1. The numbers represent the sequence of actions. C sends a join request to the server, the server transfers the current shared state of G1 to C, and C acknowledges the receipt of the state.

terface and application-specific objects or of only the user interface objects. In the latter case, the importer would be allowed either to access the application-specific objects remotely or to only view others' updates on the window.

The decision also depends on the type of updates required. For the part of the shared state that requires memoryful updates, its current state as well as the accumulated effects of the past updates on it should be transferred to a newcomer. However, for the part that requires memoryless updates, only the current state needs to be transferred.

When a client joins a stateful group, the Corona server simply transfers the current shared state of the group to the new member without disrupting the work of the existing members. An authorized client such as a group creator may also specify to the server which objects in the shared state to be transferred based on the roles of new members.

State Synchronization Service

Ideally, the updates made by a user on the shared state should be instantaneously reflected on other collaborators. However, this may be impossible in practice due to constraints such as network latency. Hence, users will inevitably experience some inconsistencies in the shared state. Hence, the developers of synchronous collaboration systems often have to make design trade-offs between a desired degree of synchronization of the shared state and the fluidity of work on the shared state the end users are allowed to perform. The tighter synchronization implies that the users would be more constrained in their work [9]. At one extreme, the users are allowed to work as freely as they like as in a collaborative brainstorming session, and no synchronization processing is performed on the shared state. At the other extreme, only one user is allowed to make updates on the shared state while the others are forced to wait. The right balance depends on the requirements of a specific collaboration situation.

In order to provide a desired degree of synchronization and control concurrent updates on the shared state of a group, the Corona server allows locks to be associated with the objects in a shared state. By default, no lock is assigned to the shared state, allowing group members to work on the shared state concurrently. While this approach may lead to chaos in general, it has been found to be more efficient and useful in many practical collaborative scenarios [9]. For example, the design of the chat box in the UARC client allows the scientists to exchange messages freely without any concern for the total ordering on messages from distinct sources. Although messages were sometimes received out of order, the scientists did not find it confusing and often commented on the effectiveness of the chat box.

When concurrent access to shared state must be synchronized, the server allows a lock to be assigned to a set of objects in the shared state. The size of a *lock set* as well as its constituents are determined by group members based on collaboration needs. An object may not belong to different lock sets. Regarding the usage of locks, a client may acquire and hold onto a lock as long as the client actively updates the objects in the lock set associated with the lock. An advantage of this lock-set based scheme is that it allows a group to specify the degree of synchronization on the concurrent operations on the shared state. The flexibility stems from the fact that a lock set may be of any desired size and that a group may have several lock sets. Since a lock set is composed of object identifiers, not the objects themselves, the locks can be managed by the server regardless of whether the shared state is centralized or replicated.

This centralized approach not only simplifies the lock management but also frees the group members from the administrative burden of lock management. Patterson, et al. [19] and Nichols, et al. [17] also discuss the merits of such a centralized serialization point for their lock management and concurrency control strategies, respectively.

Logging Service

The logging facility of the Corona server consists of two complementary services: (1) a checkpointing service for taking a snapshot of the current shared state of a group periodically or on demand; (2) a record and replay service for logging group multicast messages containing update information on the shared state. Both facilities allow a group's shared state to be persistent and outlive the group itself. The first approach is particularly useful when the group members may have to adjourn their work to a later time. In the second approach, the server effectively records an incremental change to the shared state. This allows the server to replay the updates at a later time, detailing how the shared state has reached a particular state. The record and replay capability [15] is especially useful for an end user who did not participate in a collaboration activity to find out not only the results of the activity but also how the results has been produced.

Implementation Status and Usage Examples

A prototype of the Corona server has been implemented as a multi-threaded Java application, supporting down-loadable Java applet clients. The server has been successfully tested and used in various UARC campaigns and project meetings, supporting numerous collaborative science and on-line group discussions. In two recent campaigns, approximately 40-50 participants (scientists and UARC developers) utilized our tools to conduct science on atmospheric phenomena over a three day period. The scientists were dispersed throughout North America and Europe, operating on a variety of platforms, including Windows 95, Solaris, and HP-UX with connectivity ranging from high-speed links to modems.

The initial development of the server has focused on its robustness and scalability; the server should accommodate dynamic client joins, leaves, and failures and tolerate varying network connectivity conditions. Hence, the prototype provides only the basic management services, including a group awareness service with limited role support and a state transfer service without the selective state transfer support.

Table I provides a partial overview of the client application programming interface to the Corona services. The Group Awareness/Membership interface allows clients to create, initialize, join, and leave groups. The interface also allows clients to pull awareness information such as group membership as needed. The group multicast interface allows clients to broadcast memoryful or memoryless updates on a group's shared state. The synchronization interface allows shared objects to be dynamically locked and unlocked.

The various collaborative tools discussed in Section are Java-based applets and access the Corona services through the interfaces of Table I. As of the present implementation, only the processes of the same tool form a group in the Corona server although the server itself does not impose such a restriction.

The different collaboration semantics that these tools are designed to support result in different usages of the Corona services. For example, A chat group uses the *mcastUpdateIncludeSender()* to broadcast messages so that when a new chat joins the group, it is transfered all previously exchanged messages. Message are broadcast sender-inclusively so that messages are displayed with consistent timestamps. The chat group does not utilize the synchronization service so that users may freely exchange messages.

On the other hand, a shared windows group require a tighter synchronization of its shared state. Conflicting updates on the display mode of the shared window, for example, may result in confusion. Thus the shared windows group uses the synchronization service to acquire locks on one or more shared objects before performing updates. Further, it broadcasts updates through the *mcastStateExcludeSender()* interface.

Group Awareness/Membership	*createGroup(gName,gType)*	creates a group of name *gName* of type *gType* where type is stateful, stateless, ..
	initGroup(gName,gState),	initialize a stateful group *gName* with its shared state *gState*
	joinGroup(gName, aRole)	join group *gName* with role, *aRole*
	joinAck(gName)	acknowledges the join completion of a member
	leaveGroup(gName)	leaves group *gName*
	deleteGroup(gName)	deletes group *gName*
	changeRole(oldRole, newRole)	changes the role of a member to *newRole*
	getGroupNames()	returns the names of groups currently present at the server
	getMembership(gName)	returns the membership of group *gName*
Group Multicast	*mcastStateExcludeSender(gName,*O_i *stateMsg)*	multicasts new memoryless state message for object O_i
	mcastStateIncludeSender(gName,O_i *stateMsg)*	multicasts new memoryless state message for object O_i
	mcastUpdateExcludeSender(gName,O_i*, stateMsg)*	multicasts an memoryful state update change to object O_i for group *gName*
	mcastUpdateIncludeSender(gName, O_i*, stateMsg)*	multicasts an incremental change to object O_i for group *gName*
Synchronization	*acquireLock(gName,* L*)*	acquires a lock on an object whose id is in L
	releaseLock(gName,L)	releases a lock on an object whose id is in L

Table I. Corona client application interface for group awareness and multicast services. $gState = \{(O_1, S_1), (O_2, S_2), ..., (O_n, S_n)\}$ is the set of shared state object identifiers (O_i and their byte stream state representations S_i. $L = \{O_i, O_j, ...\}$ is a set of identifiers of objects for which an identifier is to acquired or released

Related Work

In its goals, Lotus' NSTP [19] most closely resembles our Corona server. Both advocate centralized management of shared state and provide similar administrative services in its support. The semantics of shared state is client-based in both systems so that their services are generalized to a wide range of applications. Further, the notion of Place in NSTP is synonymous with the group concept in Corona, and the Things in a Place correspond to the objects in a shared state of a stateful group in Corona.

However, NSTP and Corona differ in several aspects of their shared state management support. First, NSTP does not support the notion of incremental updates. Each update on a Thing always overwrites the old value of the Thing. This limitation would make the development of tools such as our chat box or draw applet difficult as the updates on the shared states of these tools are fully incremental. One way to simulate an incremental update is to dynamically create and add a new Thing to a Place for each update. However, this would entail a difficult problem of uniquely naming new Things at runtime. Second, NSTP does not transfer shared states to clients; instead, when a client enters a Place, it is only given the names of the Things in the Place. Hence, clients always access shared objects remotely. This may significantly degrade performance in user responsiveness in a highly interactive collaboration environment, especially over a wide area network. Finally, NSTP does not support any notion of persistence in its Things or Places. On the other hand, the Corona server does not support the Facade-like capabilities for viewing the shared states of groups before actually joining the groups; users should obtain the names of groups to join externally through our session manager [14].

Coast [22] and DistEdit [12] fully replicate their shared states as well as various administrative components among client applications. As discussed earlier, a fully replicated approach may not be suitable for collaboration over a unreliable, long-haul collaboration environment such as the World Wide Web where clients may not perform administrative tasks reliably.

Suite [5], Rendezvous [20], and Jupiter [17] take a centralized approach to management of shared state. In addition to managing shared states, the Suite, Rendezvous, and Jupiter servers run application code. This may degrade the performance in user responsiveness even in a local-area network. Corona is unique among systems that take a centralized approach in that it provides the state transfer service, allowing clients to interact with local copies of shared states. This allows for high performance in user responsiveness, especially over a wide-area network.

The Corona server has provisions for both optimistic and pessimistic approaches to synchronizing shared state. By default, the server does not associate any locks to a group and allows the group members to freely interact with each other. This approach is synonymous in its intent with the optimistic synchronization approaches taken by Grove [6], Jupiter, and Coast. The difference is that our approach does not provide explicit conflict resolution mechanisms. However, the rollback or operation transformation mechanisms employed by these systems are often expensive and hard to generalize, respectively. We feel that locks are more efficient and simpler to manage when concurrent updates to the shared state should be synchronized. MMConf [3], DistEdit, and NSTP all use locks to control concurrent updates.

Many other systems provide administrative services similar in part to the services provided by the Corona server. Both IRC [18] and Zephyr [4] provide centralized messaging and notification services, which are similar to our group awareness and multicast services. However, neither of these systems has the concept of shared state or role distinction among members and is not intended to support general synchronous collaborative activities. As a transport layer subsystem, ISIS [2] supports the notion of process groups, notification of membership changes, and group multicast and may be used to build our group awareness and group notification services. However, it also lacks the notion of shared state and does not provide guarantees on delay when a client joins since the state transfer originates from another client.

Conclusions and Future Work

This paper presented our approach to providing flexible services for managing shared state in computer-supported collaboration. We believe that both the efficient development of collaborative applications and the widespread use of computer-supported collaboration technology would be greatly benefited by such services. We have identified a set of general services fundamental to shared state management and discussed different design choices possible for providing these services. The services are provided by our Corona server whose flexibility allows clients to choose what services to receive and how the services are provided based on their particular shared state management needs. An initial prototype of the server has shown the applicability of our approach through the of development of several collaboration tools.

Our current research efforts are focusing on increasing the scalability and robustness of the server and examining the issues involved in a distributed implementation of the server.

Acknowledgments

This work is supported in part by the National Science Foundation under cooperative agreement IRI-9216848.

References

[1] K. P. Birman and T. A. Joseph. Low-Cost Management of Replicated Data in Fault-Tolerant Distributed Systems. *ACM Trans. on Computer Systems*, 4(1):54–70, Feb. 1986.

[2] K. P. Birman and T. A. Joseph. Exploiting Virtual Synchrony in Distributed Systems. In *Proc. of 11th ACM Symp. on Operating Systems Principles*, pages 123–138, Austin, TX, Nov. 1987.

[3] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson. MMConf: An Infrastructure for Building Shared Multimedia Applications. In *Proc. of ACM Conference on Computer Supported Cooperative Work*, pages 329–342, October 1990.

[4] C. A. DellaFera and M. W. Eichin. The Zephyr Notification Service. In *Proc. of the USENIX WInter Conference*, Dallas, Tx, 1988. USENIX Association.

[5] P. Dewan. Flexible User Interface Coupling in Collaborative Systems. In *Proceedings of the ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 41–48, April 1991.

[6] C. Ellis, S.J. Gibbs, and G. Rein. Design and Use of a Group Editor. In G. Cockton, editor, *Engineering for Human-Computer Interaction*, pages 13–25. North-Holland, Amsterdam, September 1988.

[7] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: Some Issues and Experiences. *Communications of the ACM*, pages 38–51, January 1991.

[8] C. R. Clauer et al. A Prototype Upper Atmospheric Research Collaboratory (UARC). *EOS, Trans. Amer. Geophys. Union*, 74, 1993.

[9] S. Greenberg and D. Marwood. Real-Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In *Proc. of the Fifth Conf. on Computer-Supported Cooperative Work*, Chapel Hill, North Carolina, 1994.

[10] R. W. Hall, A. Mathur, F. Jahanian, A.Prakash, and C. Rasmussen. Corona: A Communication Service for Scalable, Reliable Group Collaboration Systems. In *Proc. of the Sixth ACM Conference on Computer-Supported Cooperative Work.* ACM Press, Nov. 1996.

[11] M. Knister and A. Prakash. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work*, pages 343–355, Los Angeles, California, October 1990.

[12] M. Knister and A. Prakash. DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors. In *Proc. of the Third Conference on Computer-Supported Cooperative Work*, pages 343–355, Oct. 1990.

[13] J. Lauwers, T. Joseph, K. Lantz, and A. Romanow. Replicated Architectures for Shared Window Systems: A Critique. In *Proceedings of ACM Conference on Office Information Systems*, pages 249–260, March 1990.

[14] J.H. Lee, A. Prakash, T. Jaeger, and G. Wu. Supporting Multi-user, Multi-applet Workspaces in CBE. In *Proc. of the Sixth ACM Conference on Computer-Supported Cooperative Work*. ACM Press, Nov. 1996.

[15] N. R. Manohar and A. Prakash. The Session Capture and Replay Paradigm for Asynchronous Collaboration. In *Proc. of the Fourth European Conference on Computer-Supported Cooperative Work*. Kluwer Academic Publishers, Sep. 1995.

[16] F. B. Schneider N. Budhiraja, K. Marzullo and S. Toueg. Optimal Primary-Backup Protocols. In *Proceedings of the Sixth International WOrkshop on Distributed Algorithms*, 1992.

[17] D. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System. In *Proceedings of UIST '95*, Pittsburgh, PA, 1995.

[18] J. Oikarinen and D. Reed. Internet Relay Chat Protocol. *RFC 1459*, 1993. Available at ftp://ds.intenic.net/rfc/rfc/1459.txt.

[19] J. F. Patterson, M. Day, and J. Kucan. Notification Servers for Synchronous Groupware. In *Proc. of the Sixth ACM Conference on Computer-Supported Cooperative Work*. ACM Press, Nov. 1996.

[20] J.F. Patterson, R.D. Hill, S.L. Rohall, and W.S. Meeks. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work*, pages 317–328, Los Angeles, California, October 1990.

[21] A. Prakash and H. Shim. DistView: Support for Building Efficient Collaborative Applications using Replicated Objects. In *Proc. of the Fifth ACM Conf. on Computer Supported Cooperative Work*, pages 153–164, Chapel-Hill, NC, Oct. 1994.

[22] C. Schuckmann, L. Kirchner, J. Schummer, and J. M. Haake. Designing Object-Oriented Synchronous Groupware with COAST. In *Proc. of the Sixth ACM Conference on Computer-Supported Cooperative Work*. ACM Press, Nov. 1996.