Corona: A Communication Service for Scalable, Reliable Group Collaboration Systems

Robert W. Hall, Amit Mathur, Farnam Jahanian, Atul Prakash, and Craig Rassmussen
Software Systems Research Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
{rhall,mathur,farnam,aprakash,rasmussn}@eecs.umich.edu

ABSTRACT

We consider the problem of providing communication protocol support for large-scale group collaboration systems for use in environments such as the Internet which are subject to packet loss, wide variations in end-to-end delays, and transient partitions. We identify a set of requirements that are critical for the design of such group collaboration systems. These include dynamic awareness notifications, reliable data delivery, and scalability to large numbers of users. We present a communication service, Corona, that attempts to meet these requirements. Corona supports two communication paradigms: the *publish-subscribe* paradigm and the *peer group* paradigm. We present the interfaces provided by Corona to applications which are based on these paradigms. We describe the semantics of each interface method call and show how they can help meet the above requirements.

Keywords

CSCW, awareness, groupware, communication services, publish-subscribe, peer group, multicast, Java

INTRODUCTION

Interest in applications that permit synchronous collaboration over the Internet has greatly increased recently. Such collaborative applications allow people located geographically apart to share and act upon data amongst themselves. This data may encompass a wide range of media and content. It may be graphical images, streams of raw data from various monitoring instruments [5], application window element state [13], files, or plain text [9, 14].

In a collaborative system, there are *data sources* or producers of the data and *data recipients* or consumers of the data. The data needs to be transported from the data sources to the data recipients over existing communication infrastructures such as the Internet where data packets may be delayed, lost, or where recipients may become temporarily unreachable. Furthermore, the recipients may dynamically join and leave the system, resulting in a changing membership of the system. Also, in certain situations, the number of recipients may be very large, and the data transport mechanisms need to deliver the data in a cost-effective manner in such situations as well.

We have been investigating such data transport issues within the context of the Upper Atmospheric Research Collaboratory (UARC) project [5]. As part of this project we are developing a collaborative system to provide space scientists with the means to effectively view and analyze data collected by various remote instruments, present ones being located in Greenland. The goal of the UARC system is to develop groupware technologies that would not only largely eliminate the need for costly trips to remote sites to collect data by providing access to remote instruments, but also to provide facilities for better and more frequent collaboration between a distributed community of scientists.

In the UARC system, data generated by remote sources such as radars, Fabry-Perot interferometers, All-Sky Imagers, and IRIS magnetometers is disseminated over wide-area networks to space scientists at their home institutions around the world (for e.g., Maryland, California, Alaska, Florida, Denmark, etc.). A "data server" gathers data from the instruments and multicasts them to clients which run at various sites around the world. These clients then graphically display the data in various data windows.

The UARC system also provides support for synchronization of data display windows among the participants. Scientists can view data in real-time through a window sharing package called DistView [13]. All changes to the shared window, caused by various window operations such as button click, pointer move, scroll, resize, etc., are immediately reflected in all copies of the window, subject to network latencies.

Figure 1 shows the typical user interface provided by a UARC client. The windows to the right and top-left display plots of data and images. Different UARC applications have access to the same data, obtained from the remote instruments, and can display different views of that data, depending on the needs of the individual scientists. A message

Computer Supported Cooperative Work '96 , Cambridge MA USA
© 1996 ACM 0-89791-765-0/96/11 ..$3.50

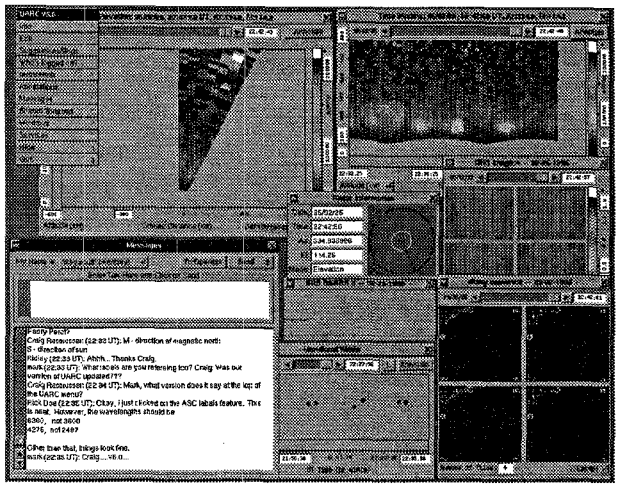

Figure 1: UARC Client Graphical Interface

window (bottom left in Figure 1) supports n-way textual talk.

Over the past two years, several prototypes of UARC have been used by scientists in their scientific campaigns with increasing effectiveness. The usage has also led to higher demands on the system in terms of tolerance to network and system unreliability and scalability to a larger number of users. Furthermore, it has become desirable to generalize the collaborative technologies developed in UARC so that they can be used for building other open, distributed collaboratories. The framework of tools that this evolving generalization effort will provide is called the "Collaboratory Builders Environment", or CBE [10].

This effort has led us to identify the need for a middleware communication layer consisting of a set of "common services" that can be used to support data dissemination and tools for collaborating using this data. One can view these services as the building blocks of a collaboratory.

In this paper we present a communication service, Corona, that provides scalable and reliable group communication facilities for such systems. The services provided by Corona are based on two communication paradigms: the *publish/-subscribe* paradigm and the *peer group* paradigm. These paradigms help define the semantics for data transport for use in collaborative systems. We present the interfaces provided by Corona to collaborative applications, describe the semantics associated with each interface method call, and show how they can meet a variety of group collaboration requirements.

The rest of the paper is organized as follows: We first describe the requirements of group collaboration systems in the "Group Collaboration Requirements" section. The next section, "Communication Services Features", describes how these requirements translate into functionality provided by a communication service. In section, "Corona Services", we describe our communication service that attempts to provide these features. Next, "Semantics of the Corona Services", describes in detail the interfaces of the Corona methods, their semantics, and how they meet the above requirements. We close with comparison to related work, conclusions, and future plans.

GROUP COLLABORATION REQUIREMENTS

In developing Corona, we considered various requirements that could be placed upon a collaborative system, with regard to the users' expectations and styles of collaboration. The requirements grew out of our experience with the initial implementation of UARC and the communication needs of the evolution of UARC, the CBE toolkit [10]. Based on this, we have identified the following set of requirements:

1. *Different kinds of data require different levels of consistency.* Data is typically replicated at each site in a collaborative system. Maintaining the consistency of the replicas usually requires that update messages to the replicated data be *ordered* consistently. In an Internet environment, however, a second determinant of data consistency is *reliable* delivery of update messages. In this paper we primarily address the reliability aspect as related to data consistency. Earlier work on ordering of messages by Greenberg and Marwood [7] and Dourish [3, 4] can be integrated with our work. Hence, in the rest of the paper, when we refer to data consistency we mean consistency impacted by the reliable delivery of update messages.

A collaborative system must be flexible with regards to the reliability guarantees it provides for data. Consider some examples from the UARC project:

- In the chatbox application, it is imperative the system reliably deliver messages to all users. Sending a message to fellow collaborators and having only some of them receive it degrades the quality and effectiveness of the collaboration.
- For scientific data obtained from remote monitoring instruments, some loss of data is usually tolerable due

to the inherent redundancy of the data. For example, the IRIS magnetometer produces a data stream in which data packets are generated at a rate of 1 every 30 seconds. The graphical viewer application that displays the data is capable of displaying it at a lesser quality of resolution if some data packets are lost.

- Within UARC, the user's view of what instruments are available should be consistent at quiescence. All users should have access to the same view of what instruments and applications are available.

The implication of this is that a communication service must not enforce a single data consistency policy for all types of data. Mechanisms must be provided to permit different consistency policies to be used with different applications and their data.

2. *Different modes of collaboration require different kinds of awareness.*
Users collaborate in a different manner depending upon the nature of the task and data being collaborated upon. These different modes of collaboration lead to different *awareness* requirements described below. By awareness, we mean information pertaining to who the users are, what groups they belong to, and what access they have to the data.

- Users collaborating in shared workspaces need to know with whom they are sharing the workspace and what operations are being performed and by whom. Users of a shared data annotation window, for instance, need to know with whom they are viewing the window and who is annotating the data. Additionally, users need to be made aware of failures. A user may wish to take certain actions if one of their collaborators loses connection to the collaboratory.

- Users viewing streams of instrument data may be less concerned as to who is looking at the data as they may wish to think about it in private.

Thus a communication service needs *flexibility* in maintaining and providing user access to knowledge of other users and failures. Not providing a means for applications to either request membership and failure information or receive updates dynamically weakens the quality of collaboration available to the user who could benefit from this knowledge.

3. *Dynamics in consistency and awareness requirements vary upon usage.*
A user's awareness of other users of a particular collaborative application, as well as the consistency requirements on the data, are not static during the course of a collaboration. In particular:

- *On-demand awareness* is needed to provide awareness of whether a particular user or set of users are looking at a particular data when that data set becomes the center of attention. Normally, such awareness is not necessary. For example, users of a UARC instrument data viewer may be viewing an atmospheric image but not be concerned about who else is viewing it. When a particular user starts annotating that image, though, the users need to be aware of who is annotating and seeing that annotation.

- Data may have to be delivered more reliably and consistently when it is the subject of discussion. Continuing with the data image example, missing data in an image or graph would not be tolerated when it is the focal point of attention among users.

Thus, a communication service must be capable of dynamic change in the nature of the consistency of data and membership knowledge of a collaborative group.

4. *Different participant roles imply different awareness requirements.*
Within a collaboration, some users may have a higher priority than others. There may be users that have the privilege of modifying the data. We refer to such high-priority users as principals. Other users, referred to as *observers,* may only be allowed to receive data. For example, in a collaboration involving a whiteboard application, the principals could write on the board, while the observers could only passively view the changes.

Principals in a collaboration need a high level of awareness of each other during the session since each can potentially make modifications to the shared data. On the other hand, little or no (as well as less accurate information) may be quite acceptable to observers. Observers need not be notified of failures or leaving or joining of other participants. Thus, a communication service must be capable of supporting different classes of users within a single collaboration.

5. *Different collaborative applications have different scalability requirements.*
Some collaborative applications by nature are oriented towards small groups of users, while others are oriented towards large groups. For example, in UARC, a single data source application such as the radar instrument, needs to disseminate data to potentially hundreds, even thousands of users. On the other hand, a DistView shared window [13], typically has tens of users, often much fewer. The communication service must support both kinds of applications.

6. *Failure notification.* Users need to be notified promptly of failures, particularly when strong awareness of group members is necessary. For example, a failure that causes some users in a shared window collaboration to lose connection to the collaboration, should be announced to all surviving users.

COMMUNICATION SERVICE FEATURES

The goal of a communication service is to deliver data from a source to one or more destinations in a manner that meets the semantic requirements of the data being delivered. In the Group Collaboration Requirements section we described a variety of such semantic requirements. Further, the communication service must be able to operate over existing communication infrastructures such as the Internet which are subject to congestion, packet loss, and transient partitions. To address the requirements of collaboration systems and the operating environment, the communication service must be concerned with four main features: reliability, awareness, failure-notification, and scalability. We discuss each below.

Reliability

The need for *reliability* stems directly from the fact that the communication service needs to function in environments such as the Internet where data packets may be delayed, lost, and destinations may become temporarily unreachable in the system. Further participant processes may also fail.

In collaborative systems, processes need to reliably transmit messages from a source to multiple destinations (also referred to as a reliable multicast). This is a significantly more difficult problem from the problem of reliably sending messages between two processes (and for which TCP has been so successful over the years). This is so for the following two reasons. Firstly, network congestion and transient partitions may cause destinations to become temporarily unreachable. This prevents a multicast of a message to multiple destinations from terminating successfully: the message is received by destinations that are reachable, but not by those that happen to be unreachable at that time. Secondly, it is possible that the sender of a message may crash after it has multicast one or more messages to multiple destinations, where some of the destinations have received all of the messages, but others have not.

Awareness and Failure Notification

Depending on the nature of the collaboration, the users have varying degrees of *awareness* of other participants of the collaborative session. We denote by *view* the set of users that a given user is aware of [1]. As users dynamically join and leave the collaborative session or if users are perceived to have failed, the awareness information maintained for the users needs to be updated carefully. Similarly, when user site failures are detected, appropriate failure-notification needs to take place.

Scalability

Certain data streams in the collaboration system may need to be delivered to potentially large numbers of users, and the communication service should allow for this in a cost-effective manner, i.e., communication protocols for certain data streams must be *scalable*.

By scalability we mean that the proposed protocols for data delivery are cost-effective even when there are a very large number (100's, 1000's, even tens of thousands) of destinations that the data needs to be delivered to. The cost metric that we use to determine the scalability of the protocols is primarily number of messages. Other metrics such as latency are possible and we plan to investigate these in the future.

CORONA SERVICES

In order to support the features described in the Communication Services Features section, we propose a communication service, Corona. Corona provides two classes of services: the *publish/subscribe* service and the *peer group* service. The publish/subscribe service is based on the *publish/subscribe* paradigm [11], while the peer group service is based on the *peer group* paradigm.

The publish/subscribe paradigm is characterized by one or more data sources or *publishers* sending data to multiple recipients or *subscribers*. The form of communication here is of an *anonymous* nature. The publishers are aware of the set of subscribers, but the subscribers are unaware of each other and are only aware of the publisher that they are receiving data from. The publish/subscribe paradigm supports a weak form of reliability and awareness for the subscribers, as we will describe in the next section. This form of communication, however, lends itself to scalability, as we will also show in the next section.

In order to implement this paradigm, we propose a two-level architecture, illustrated in Figure 2. In this architecture, a publisher multicasts data to a set of intermediate nodes, referred to as *distributors*. The distributors then route the data to other distributors, which in turn send the data to the local subscribers.

A key point of this architecture is that by introducing the distributors we are *minimizing system-wide awareness and change*. A distributor is aware of *only* its local set of subscribers and has *no* awareness of any of the other subscribers in the system. This allows the system to scale nicely as subscribers can join and leave without affecting the entire system. Furthermore, since distributors act as routers and multicast the messages to the local subscribers only, the burden of a single source multicasting a message to a very large number of destinations and dealing with the associated ack-implosion is alleviated to a large extent.

The second type of service provided by Corona is the *peer group service*. It is based on the *peer group paradigm* of communication which is characterized by all of the group members being aware of each other. This allows each member to send and receive data to and from each other. Thus communication is not anonymous, rather it is *named*. Thus in a *peer group*, all members can be both a data source and a data recipient, and are aware of each other. The peer group paradigm thus supports a strong form of reliability and awareness for

[1] This usage of the term *view* is distinct from its usage in user interface work.

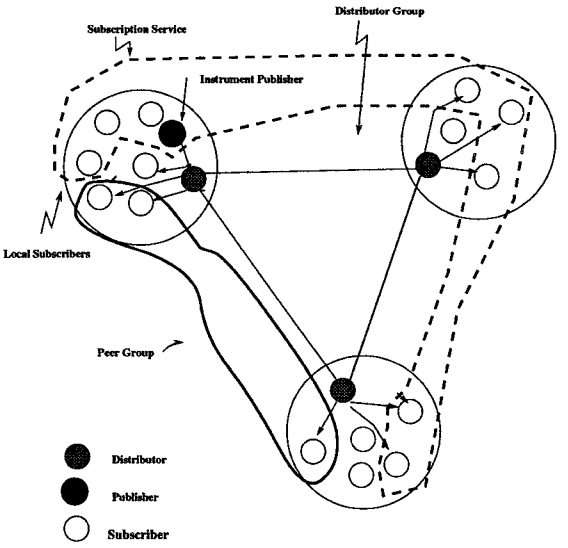

Figure 2: System Architecture

	Publish/Subscribe Service	Peer Group Service
Paradigm Idea	Anonymous communication	Named communication
Reliability	Weak	Strong
Awareness	Weak	Strong
Failure Notification	Minimal	Extensive
Scalability	Yes	No

Table 1: Comparison of the two classes of services provided by Corona

the group members.

In a peer group, it is possible to have certain members as principals and other members observers. The principals are provided with strong reliability and awareness guarantees, while the observers have much weaker guarantees. Further, observers can only receive data while principals can send and receive data. We elaborate on this in the next section. Corona allows for a subscriber to join an existing peer group. This allows a user to dynamically alter their awareness and reliability guarantees.

In Table 1, we summarize the key aspects of the two classes of services provided by Corona.

SEMANTICS OF THE CORONA SERVICES

We describe the semantics of the publish/subscribe and peer group services provided by Corona. In order to justify some of the design choices made, we first analyze the cost of implementing the semantics.

Reliability/Awareness and Scalability Tradeoffs

Reliability and awareness are closely related to each other and both have an inverse relationship with scalability. Strong forms of reliability and awareness are costly to implement in terms of number of messages, and hence less scalable. Thus there is a fundamental tradeoff between the reliability/awareness requirements and the scalability requirements. Here we formalize this tradeoff. This will then allow us to determine appropriate forms of reliability/awareness for modes of collaboration that need to be scalable and those that do not need to be scalable.

Assume that there are n users in the system. We evaluate the cost of ensuring reliability/awareness when a new user *joins* or *leaves* the system, and when a process is suspected of having *failed.*

Ensuring awareness amongst the group of processes of a user join/leave/fail can be achieved using a coordinator based solution. The coordinator first proposes a view to the n users and waits for acknowledgments (ack's) from them. The coordinator then collects the ack's, revises the proposed view if necessary, and commits the view to the users in the revision. The number of messages sent in order to ensure awareness of the user join/leave/fail is thus of the order of $3n$.

Reliability can also be ensured in a similar manner. The sender multicasts a message, the receivers ack the message to the sender, the sender sends out a *stable* message to all, allowing them to agree and discard messages from buffers. Thus the message cost of ensuring reliability and awareness are both of the order of $3n$ messages.

Publish/Subscribe Service Interface and Semantics

The publish/subscribe service consists of one or more publishers disseminating data through the distributors to sub-

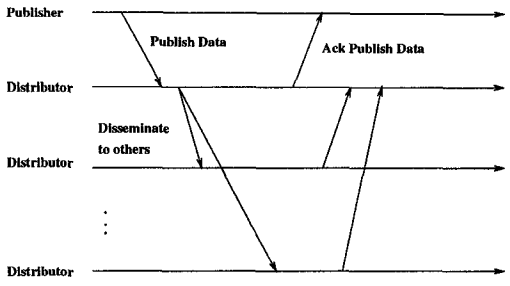

Figure 3: Publish Data for a Service

scribers to the data. The actual interface provided by the publish/subscribe service is presented in Table 2. We describe the semantics of the important methods below.

createService(in String serviceName, out ServiceId aService)
A publisher calls this method to create a service with the given name. We define a *service* as a stream of messages from a publisher to the set of subscribers. The service identifier, aService, is used to identify the service in calls to other interface methods. The new service is registered with a single distributor, referred to as the *primary distributor*. The primary distributor then shares this information with the other distributors.

publish(in ServiceId aService, in Message aMsg)
This method is used to publish data for a service. The sequence of events are illustrated in Fig. 3. The publisher first sends the message to its primary distributor. The primary distributor then sends the message to all of the other distributors. Each of these distributors then sends the message to the local subscribers.

subscribe(in ServiceId aService, in RoleType role)
A subscriber sends a subscribe message to its primary distributor. The primary distributor adds this subscriber to its subscriber view. Any data messages delivered to the distributor for this service will now be sent to this subscriber. A publisher would also use this interface to establish themselves as a publisher to a service.

unsubscribe(in ServiceId aService)
A subscriber sends an unsubscribe message to its primary distributor. The distributor removes this subscriber from its view.

Reliability/Awareness Cost in the Publish/Subscribe Service

The publish/subscribe service ensures a strong form of reliability and awareness only amongst the distributors and not amongst the subscribers, who are the end-users of the service. It is possible for the subscribers to experience occasional gaps in the data they receive. This may happen if the distributor that the subscriber is connected to crashes. The subscriber will then re-connect to a new distributor and will start receiving messages from that distributor. It is possible that the new distributor may have the messages that the subscriber may have missed, but it cannot be guaranteed.

The cost to achieve this weak form of reliability and awareness is of the order of $3d$, where d is the number of distributors. If we had tried to ensure this level of reliability and awareness amongst all the subscribers in the system, then the cost would be significantly higher, since the number of subscribers can be significantly larger than the number of distributors. The rationale for this choice is the observation that many data sources (e.g. instrument data) only require weak forms of reliability.

Peer Group Service Interface and Semantics

For the peer group service, the interface methods, presented in Table 3, and their associated semantics are discussed below. Note that the peer group service supports two types of members– observers that have limited peer group awareness and can only receive data, and principals who have strong awareness of group members and can send and receive data.

createGroup(in String groupName, out GroupId aGroup)
This call allows a user to create a peer group and assign a name to it. The distributors maintain information about existing peer groups in the system.

joinGroup(in GroupId aGroup, in RoleType role, in callback AwarenessUpdate)
A member can join a specific peer group as a principal or as an observer in the group, specified by the 'role' parameter. (Note that access control as to who can use this interface to join as a principal or observer is a policy of a higher-level application, such as the CBE Session Manager [10].) A principal is at all times aware of other principals in the group and also receives all failure notifications, delivered to the application via the callback routine parameter 'AwarenessUpdate'. An observer does not receive view information related to the members of the peer group. It can, however, query the distributor to obtain this awareness information, using the interface methods getGroupList() and getMembersOf(group), shown in Table 3.

leaveGroup(in GroupId aGroup)
A peer group member leaves a group by executing this method. The primary distributor removes this peer from its view, and if the peer was a principal, the other distributors are notified so the peer can be remove from their views, as well as notifying principals of this group of the change.

groupMulticast(in GroupId aGroup, in Message aMsg)
A principal sends a message to the peer group members using this method. Strong reliability guarantees are provided on the delivery of this message to the other principals in the group, while the observers have weaker reliability guarantees (they can miss certain messages).

Method	Function
connect(in Address serverHost, in int serverPort, in int localPort)	connect to distributor
disconnect()	disconnect from distributor
createService(in String serviceName, out ServiceId aService)	create service
deleteService(in ServiceId aService)	delete service
subscribe(in ServiceId aService, in RoleType role)	subscribe to service (as publisher, subscriber)
unsubscribe(in ServiceId aService)	unsubscribe from service
publish(in ServiceId aService, in Message aMsg)	publish data for a service
getServiceList(out ServiceIdList services)	get list of available services
getGroupList(out GroupIdList groups)	get list of available peer groups

Table 2: Corona Publish/Subscribe Service Interface

Method	Function
connect(in Address serverHost, in int serverPort, in int localPort)	connect to primary distributor
disconnect()	disconnect from distributor
createGroup(in String groupName, out GroupId aGroup)	create peer group
groupMulticast(in GroupId aGroup, in Message aMsg)	multicast a message to the group
joinGroup(in GroupId aGroup, in RoleType role, in callback AwarenessUpdate)	join a peer group
leaveGroup(in GroupId aGroup)	leave a peer group
getGroupList(in GroupIdList groups)	get list of available groups
getMembersOf(in GroupId aGroup, in RoleType role)	find out membership of group

Table 3: Corona Peer Group Interface

Reliability/Awareness Cost in the Peer Group Service

The peer group service ensures strong awareness and reliability amongst the principals. The cost to achieve this is of the order of $3p$, where p is the number of principals. Since the number of principals in a peer group is typically small, this cost will be reasonable.

Example Usage Scenarios

We describe below some scenarios that illustrate how the above interfaces can be used by collaborative applications. The first collaboration scenario pertains to a group where a source is disseminating data to users who view the data but do not interact or change this data. We describe how a publish-subscribe group is created, subscribed to, and published to by publishers and subscribers. The second collaboration scenario pertains to a collaboration group where participants can collaborate on the data, interacting and changing the data. We describe how a peer group is created, how members join it, messages are sent, and view changes and failures are handled.

Within the context of UARC, we describe a publish-subscribe group for the IRIS magnetometer data. The magnetometer data source application (the publisher) first uses the connect() call to establish a connection to its primary distributor. After connecting, the publisher uses the createService() method to create a new service, "iris-data". From this call it gets back an identifier for this service. The primary distributor of this publisher then publicizes the new group to other distributors. Users of client applications, such as the UARC 2-D data viewer for viewing the data, would join the collaboration by first using connect(), then the getServicesList() method to get a list of known services. From this list, they would choose the identifier for "iris-data" and use subscribe() to join "iris-data" in the role of subscriber. Note that since this is a publish-subscribe group, other subscribers and publishers would be unaware of such joins. The use of a publish-subscribe group is reasonable for this application, because the users are privately viewing data.

We next describe a scenario where a peer group is formed for the purpose of annotating the IRIS data. Assume that the subscribers of the IRIS service decide to form a peer group to annotate the IRIS data displays. A user application can use the createGroup() method to create the peer group "iris-annotation". The primary distributor will distribute this new group information to all distributors. After connecting and getting the list of groups, users can join the collaboration by executing joinGroup(). They can choose the role of a principal—they will be able to create, edit and delete annotations, or the role of an observer, capable of only seeing the annotated data, but not modifying it. Who can join as a principal and who can join as an observer is left to the using application. When another principal joins (or leaves the group by executing the leaveGroup()), the distributors will notify all other principals in "iris-annotation" via the awareness notification mechanism using each principal's AwarenessUpdate callback routine provided in joinGroup().

Note that the awareness levels are reasonable for this application, because annotating is an interactive task, and each

principal will want to be aware of others annotating the data. If an observer failed or left, the other observers in "iris-annotation" would not be notified, nor would the principals, unless they requested such notification.

How Corona Addresses Group Collaboration Requirements

We now return to the original group collaboration requirements outlined in the Group Collaboration Requirements section and describe how they are addressed by Corona.

1. *Different kinds of data require different levels of consistency.* Corona provides weaker consistency requirements and stronger consistency requirements with the appropriate semantics through the publish/subscribe and peer group services, respectively.

2. *Different modes of collaboration require different kinds of awareness.* Corona provides weak awareness guarantees to peer group observers and subscribers and strong awareness guarantees to peer group principals by distinguishing between them through differences in view management.

3. *Dynamics in consistency and awareness requirements vary upon usage.* Corona provides for dynamic change in reliability and awareness guarantees through the publish/subscribe service's interface for allowing subscribers to join peer groups as observers and thus gain stronger awareness and vice-versa.

4. *Different participant roles imply different awareness requirements.* By providing the distinction between principals and observers within a peer group collaboration, Corona provides different awareness guarantees for each type of peer group member.

5. *Different collaborative applications have different scalability requirements.* Corona provides data dissemination capability to large groups with weak reliability and awareness guarantees with the publish/subscribe service and peer shared data capability to small groups with strong reliability and awareness guarantees with the peer group service.

6. *Failure notification.* Corona provides failure notification to principals in a peer group collaboration as part of the strong awareness guarantees provided to peer group principals.

CORONA PROTOCOLS AND IMPLEMENTATION

We describe briefly the protocols used to implement the primitives of the two classes of Corona services. There are two main aspects to the protocols: *view management* and *multicasting*. View management ensures awareness of the various users as well as performs failure notification. Multicasting involves sending the message to one or more destinations and ensuring reliable delivery.

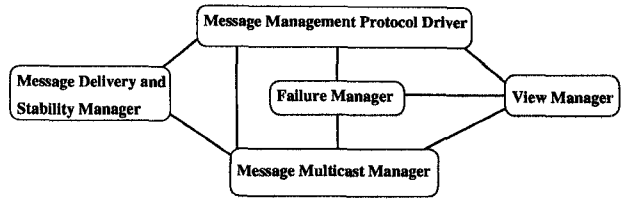

Figure 4: Major Components of the Software Architecture

The main components used to implement the protocols are illustrated in Figure 4. We describe each and their implementation below:

View Manager

The view manager component is responsible for ensuring that a given user is aware of other users in the collaborative sessions it is a part of. Depending on the role of the user in these collaborative sessions, i.e., whether the user is a publisher, subscriber, observer, or principal, the view manager executes the appropriate protocols for ensuring that the user maintains the level of awareness associated with its role.

Each distributor process maintains three views: the view consisting of the other distributors in the system; the view consisting of the set of local publish-subscribe clients; and the view of peer groups and their members. In the distributor implementation, these views are managed by three components—a DistributorViewManager that uses a strong group membership protocol [8] to maintain a consistent view of active distributor processes; the PublishSubscribeClientViewManager which maintains a single distributor's view of services and local publisher/subscriber clients, and the PeerGroupViewManager which maintains a view of known peer groups and their members–observers and principals.

Message Multicast Manager

This component multicasts a message to a group of users. It obtains from the ViewManager a list of destinations, and for each destination, the timeout and retransmit values from the Failure Manager. The message is then sent to each destination and retransmitted at the *timeout* interval or until an acknowledgment is received. If the number of retransmits exceeds some maximum, the Failure Manager is notified.

Failure Manager

The Failure Manager has several functions. It maintains timeout and retransmit values for each destination that the user wishes to send messages to, providing this information to the Message Multicast Manager when it needs to multicast a message to a group of users.

The Message Multicast Manager in turn informs the Failure Manager when it does not receive an acknowledgment for a particular message from a destination. The Failure Manager then informs the appropriate view manager to initiate a view

change that would exclude the particular destination from the view of this process. A further function, failure notification, is carried out on behalf of the View Manager. For instance, suppose a principal in a group is detected to have failed. After being notified of the failure, the View Manager provides the Failure Manager with a list of active principals for that group. The Failure Manager then sends a failure notification to all active principals.

Message Delivery and Stability Manager

This component is responsible for delivering received messages to the application. It also buffers the message until it knows that all of the other users in the group have received it. It thus helps to ensure the various forms of reliability supported by the services. Ordering (such as FIFO) and other delivery rules are applied to messages here.

Message Management Protocol Driver

This component interprets each incoming message and interacts with the appropriate component (View Manager, Message Multicast Manager, Failure Manager) to take further actions based on that message type and content.

Implementation Status

The initial implementation of Corona provides the publisher/subscriber and peer group client libraries that provide the interfaces itemized in Tables 2 and 3. Support for view maintenance, reliable UDP-based multicast, failure detection, and awareness notification is provided by the distributor. This C++ implementation has been tested on UNIX platforms including SunOS 4.2, Solaris, HP-UX and NeXT.

The development of the Java-based CBE [10], which has a Web-browser-based *applet* architecture of a Session Manager coordinating multiple applets, each being a node in an applet group (a distributed chatbox, data viewer, shared editor, or other collaborative application), is motivating the Corona Java implementation. Corona will provide applet group communication and awareness services by allowing Java applets to communicate over the Internet via Corona's distributors. The Java implementation of Corona that is under way to support the CBE consists of the client publish-subscribe interface, client peer group interface, and reliable multicast transport layer that enables Java applet clients to communicate with existing C++ distributors.

In both the C++ and Java implementation, each component consists of one or more self-contained classes with well-defined interfaces. The Java implementation is being developed on Solaris, and will likely be used on other platforms that support Java. For more information on the Java implementation, see the Corona Web Page at http://www.eecs.umich.edu/~rhall/corona.html.

RELATED WORK

Many communications services providing group membership and multicast have been developed, including Isis [2], Consul [12], Transis [1], Horus [15], and SRM [6]. Our service, Corona, which is strongly motivated by the unique needs of group collaboration systems, is different from these services in a number of respects. Corona differentiates amongst classes of users, and provides different users with different levels of reliability and awareness information. In earlier communication services, all users received the same levels of reliability and awareness information. Further, in Corona, reliability and awareness are traded-off with scalability. So the peer group service in Corona provides strong levels of reliability and awareness information in a non-scalable way, to be used in small, synchronized groups, and the publish/subscribe service provides weaker reliability and awareness information, in a scalable way, to be used for large-scale dissemination of data. Corona also supports dynamic transitions between the two classes of services.

CONCLUSIONS AND FUTURE WORK

In this paper, we have considered the problem of providing communication protocol support for large-scale group collaboration systems for use in environments such as the Internet which are subject to packet loss, wide variations in end-to-end delays, and transient partitions. We identified a set of requirements that are critical for the design of group collaboration systems. We presented a communication service, Corona, that attempts to meet these requirements.

Corona supports two communication paradigms: the *publish/subscribe* paradigm and the *peer group* paradigm. In the publish/subscribe paradigm one or more data sources or *publishers* send data to multiple *subscribers*. This paradigm is characterized by the *anonymous* nature of communication, where a publisher is aware of the service and the possibility of a set of subscribers, but the subscribers are unaware of each other and only aware about the service that they are receiving data from. In the peer group paradigm of communication on the other hand, all the group members are aware of each other, and can send and receive data to and from each other.

We presented the interfaces provided by Corona to group collaboration applications allowing them to utilize the above communication paradigms. We described the semantics associated with each interface method and showed how it is able to meet the requirements as identified. We described the functionality of the components that provide these semantics and their implementation status.

Future plans include integration with the CBE applets for use by UARC's scientists, exploration of further communication services, generalization of the two-level hierarchy to n-levels, and exploration of various multicast semantics.

ACKNOWLEDGMENTS

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

REFERENCES

1 Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A

Communication Sub-System for High Availability. Technical Report TR CS91-13, Computer Science Dept., Hebrew University, April 1992.

2 K. P. Birman. The Process Group Approach to Reliable Distributed Computing. *Comm. of the ACM*, 36(12):37–53, Dec. 1993.

3 P. Dourish. Developing a Reflective Model of Collaborative Systems. *ACM Transactions on Computer-Human Interaction*, 2(1):40–63, 1995.

4 P. Dourish and V. Bellotti. Awareness and Coordination in Shared Workspaces. In *Proc. of the Fourth ACM Conference on Computer-Supported Cooperative Work*, Toronto, Canada, October 1992.

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

6 S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proc. of the ACM SIGCOMM Symp.*, pages 342–356, Cambridge, MA, Aug. 1995.

7 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.

8 F. Jahanian, S. Fakhouri, and R. Rajkumar. Processor Group Membership Protocols: Specification, Design, and Implementation. In *Proc. of Symp. on Reliable Distributed Systems*, Princeton, NJ, Oct. 1993.

9 M. Knister and A. Prakash. Issues in the Design of a Toolkit for Supporting Multiple Group Editors. *Computing Systems – The Journal of the Usenix Association*, 6(2):135–166, Spring 1993.

10 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.

11 A. Mathur, R. Hall, F. Jahanian, A. Prakash, and C. Rasmussen. The Publish/Subscribe Paradigm for Scalable Group Collaboration Systems. Technical Report CSE-TR-270-95, The University of Michigan, Ann Arbor Michigan, November 1995.

12 S. Mishra, L. L. Peterson, and R. D. Schlichting. Consul: A Communication Substrate for Fault-Tolerant Distributed Programs. *Distributed Systems Engineeering Journal*, 1(2):87–103, Dec. 1993.

13 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.

14 M. Roseman and S. Greenberg. GROUPKIT: A Groupware Toolkit for Building Real-Time Conferencing Applications. In *Proc. of the Fourth ACM Conf. on Computer-Supported Cooperative Work*, pages 43–50, Toronto, Canada, October 1992.

15 R. van Renesse, T. M. Hickey, and K. P. Birman. Design and Performance of Horus: A Lightweight Group Communications System. Technical Report TR94-1442, Computer Science Dept., Cornell University, Aug. 1994.