Group Management and Communication Support for Collaborative Applications

Erik Wilde (wilde@tik.ethz.ch)
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH Zürich)
CH – 8092 Zürich

Abstract

In this paper, an architecture for communication support for collaborative applications is described. The motivation for the design of this architecture is the observation that generic support for group communications is an area which received not much attention until now. The design is based on two components, a *group management system (GMS)* and *group communication support (GCS)*. The GMS is responsible for managing the name space of the support platform. Users and groups are the two entities of the name space, and two different relationships between them (membership and manager) can be established. This way it is possible to reflect the structure of collaborative workers inside the GMS. The GCS component is responsible for establishing connections between collaborative applications using the GMS/GCS and for hiding the details of the multicast transport infrastructure from the application. It is possible to bind users and groups to specific applications and multicast transport services. This way any group can be used by different applications using different transport services. The main advantages of GMS/GCS are reduced implementation costs, a shared name space of users and groups, and a simple interface to different multicast transport services.

1 Introduction

Over the past ten years, computers have evolved from a personal tool, which is used mainly by isolated individuals who exchange data occasionally, to a platform which is used by groups of users for collaborative tasks. The so-called "group orientation" described by Veríssimo et al. [27, 28] can be seen as the new paradigm for distributed systems. However, between the low-level communication facilities (such as broadcast and multicast facilities) and high-level toolkits (for special types of applications) there is a gap that has to be filled with a basic communication support facility for collaborative applications.

The advantages of providing such a support environment are mainly in three areas. The first advantage is, that collaborative applications would be independent from the underlying communication facilities, making it easy to exchange them with other communication platforms. In the second place, implementation time could be reduced by using an abstract support platform. Thirdly, interworking of collaborative applications would be made possible by sharing the same database of users and groups which is managed and provided by such a platform.

This paper, which describes the design of such a support platform, is structured into six more sections. The next section will give an overview over related work. Section 3 will give a short description of collaborative applications and their requirements, leading to a description what is needed to support them. These requirements are then used to construct an architecture of a support platform for collaborative applications, which is described in section 4. Section 5 gives examples on how this architecture may be used for the implementation of new collaborative applications. Some issues which are of further interest, but not addressed in the current design, are sketched in section 6. Finally, section 7 gives the conclusions of this paper.

2 Related work

Work on support for group communications began in the late eighties. An early example of such a platform is the multicast interface for Unix 4.3 described by Hughes [11]. The interface is rather simple, offering eight primitives for multicast communications over Unix sockets. It is possible to *create* identifiers and to *join* and *leave* groups. However, the management of these identifiers and the distribution of this information are not within the scope of this interface.

The group system described by Bannon and Page [2] is much more suited to support collaborative applications. It supports an elaborated group construction concept which includes administrators, members, and guests. It is also possible to limit groups by the number of members and possible participants. Group nesting is possible, ie a group may contain other groups. However, group offers no communication support, thus leaving it up to the applications to find out which transport infrastructure can be used and how to use it. Therefore, its usefulness is limited to one aspect of the support of collaborative applications.

*x*AMp by Rodrigues and Veríssimo [22] is a group communications service which offers configurable multicast communications. The *x*AMp service relies on an abstract network service which is called an unreliable multicast service. This service is then used to provide group communication facilities. While *x*AMp focuses on the provision of multicast communications with guaranteed properties, it is not very strong with respect to addressing. Group addresses are just lists of unicast addresses. These lists may also be used as logical names, with join and leave operations to add addresses to and remove addresses from the lists. However, the support for abstract names and name management is rather weak.

The CoDraft application and its underlying multi-party communication platform as described by Kirsche et al. [15] is a good example for the development of a support platform for collaborative applications. CoDraft, a shared sketching tool, was first implemented on top of point-to-point communication services. Since it became clear that most of the data transmitted had to be transferred to all other participants of a sketching session, a multi-party communication platform consisting of four services was invented. These services are an application-specific group management service, a multicast distribution service, a multi-party file transfer service, and a voting service. Although the CoDraft multi-party communication platform offers many functions required for the support of collaborative applications, it is restricted to one transport service. Thus is it not possible to use the variety of multicast services which are available on today's network architectures.

Heijenk et al. [9] describe a service for multimedia multi-user applications which is based on the ATM adaptation layer (AAL). The AAL functionality is extended with multimedia and multi-user communication facilities. This model also concentrates on the service elements and does not provide application support for abstract names or groups. Multi-user connections are opened by using a call establishment service with a list of addresses of participating users. Furthermore, this model is restricted to ATM multicast environments.

Group communication support for distributed multimedia applications is the framework of the GCommS project described by Mauthe et al. [17]. This project has three main goals, to investigate group management, to make possible the specification of application requirements (using quality of service characteristics), and to find a transport infrastructure which is sufficient to support these services. However, since this project is in a relatively early stage, it is not possible to give a final assessment.

3 Requirements of collaborative applications

Most of today's collaborative applications are based on top of proprietary communication architectures (which are in turn based on standard communication architectures such as TCP/IP). The reason for this development is the lack of high-level support for collaborative applications. There is a large gap between interfaces to multicast transport services such as IP multicast as described by Deering [5], XTP multicast [26, 29], or ATM multicast [1], and very high-level abstractions such as Coordinators introduced by Depaoli and Tisato [7, 8], or GroupKit, a toolkit described by Roseman and Greenberg [23]. The goal of this paper is to present a model which may fill this gap.

IP or ATM multicast are good platforms for the development of collaborative applications (much better than unicast infrastructures which are also used frequently), but using them without additional support costs a lot of effort to be put in the management of groups and often limits the application to the particular transport platform selected for the implementation. Furthermore, it limits the collaboration model (ie the groups that user may form to get tasks done) to the respective application. On the other hand, complex CSCW toolkits are often rather complicated, and, more important, too restrictive to be used in collaborative applications in general. This is caused by the toolkits' aim, which is to offer more than pure technical communication support, ie coordination (or collaboration[1]) support. For a limited set of applications (depending on the design of the specific toolkit) this high-level support is useful. But because of this specialization, these toolkits are not usable as a general platform for the development of collaborative applications. Consequently, they can not be used as a common base for collaborative applications that would enable these to share information about users and groups of users.

[1]The terms coordination, collaboration, cooperation, and communication are very hard to distinguish, especially when literature from computer science (communication systems) and CSCW (which is interdisciplinary) is mixed. We will use the term communication in the technical sense. For a discussion of these (and more) "c-words" see Kling's article [16] about CSCW.

What is needed as a general support platform for collaborative applications is a model which is powerful enough to provide the applications with services that are more useful (in terms of abstraction) than standard communication system support (ie interfaces to multicast transport services), but does not impose any limitations with regard to communications on them. This approach has three major benefits:

- Using such a support platform for collaborative applications, the implementation effort for new applications would be reduced, because functionality needed by all collaborative applications would only have to be implemented once.
- Because such a model would be usable by any collaborative application, these could share a common support platform. Being built on top of this common platform would also mean sharing the data which is made available by this platform, such as user groups and addresses of users and groups. Consequently, cooperation among the collaborative applications would be made possible.
- Using a general support platform would hide the differences of underlying transport services from the applications. For example, sender-initiated and receiver-initiated multicast protocols (for an overview of their behavior see the article by Pingali et al. [19]) have a quite different way of addressing groups of users and establishing connections to existing groups. These differences could be hidden by the support platform for collaborative applications.

Based on these assumptions, it is necessary to analyze the requirements of collaborative applications. Some work on this field has already been done, eg the articles by Rodden et al. [20, 21], or the article by Heijenk et al. [9]. However, most of the literature is about support for specific applications or application types.

The following sections focus on different areas of communication support for collaborative applications. Section 3.1 deals with names, ie the abstract entities which are used to identify collaborating users and groups of users. Because names are an abstraction, section 3.2 discusses the topic how the mapping of names to addresses may be achieved. Finally, after having addressed communication partners, it is necessary to transmit data back and forth. Section 3.3 describes which patterns of data transfer can occur between the participants of a collaborative task.

3.1 Name management

In order to identify users and groups of users regardless of their physical address, it is necessary to give them names. For a short discussion of names and their relation to addresses, see the RFC by Saltzer [24]. Using names, it is possible to identify entities in an abstract way. Names are therefore location-transparent, ie if a user changes his address (perhaps when changing from one location to another, or simply when logging in to another machine), the name does not change. Because the number of potential users should not be limited by a support platform for collaborative applications, the name space has to be structured to allow the creation of domains, ie logically structured name spaces. One possible way to structure a name space is described by the ITU directory standards [12], which use a tree structured name space (the so-called *directory information tree (DIT)*) where each node of this tree is uniquely identified by a *distinguished name (DN)*. In this approach, each branch of the tree constitutes a *relative distinguished name (RDN)* which, followed and concatenated from the root to any object, give the DN of this object. The RFCs by Kille [13, 14] describe the possibilities of directory naming in more detail.

However, even if names of users and groups can be used, there is still some functionality missing which makes it possible to from groups of users and to change these groups during their lifetime. These functions must provide means to create and delete groups, to join and leave groups, and to restrict these actions to an authorized set of users. Consequently, naming in the context of collaborative applications also includes group management functions, which must be general enough to implement any management strategy a particular application wants to use. Typical scenarios in collaborative applications are group of users in a collaborative activity who dynamically form subgroups to work on special topics. The creation, modification, and deletion of these subgroups must be supported by the platform. Furthermore it must be possible to control these groups with respect to size (how many members are possible), member types (ie are groups allowed to be members), and access policies (who is allowed to join a group, and are there any members who must confirm a join request). The possibility of nested groups, ie groups within groups, leads to the concept of direct and indirect memberships. A user is a direct member of a group when he is listed as a group member. He is an indirect member of a group if he is member through one or more levels of indirection (ie member of a group which is a member of the group in question).

3.2 Addressing

Addresses are identifiers which are used to locate entities. According to a common definition first given by Shoch [25], "the name of a resource indicates what we seek and an address indicates where it is". Applying this definition to the support platform for collaborative applications, it becomes clear that the handling of addresses should be left to the support platform, while application can use names, which offer the greater level of abstraction. However, because addressing schemes are quite different in different multicast transport services, it is necessary to have a closer look on this issue.

Multicast addressing can be divided into two classes, sender-initiated addressing and receiver-initiated addressing. For some discussions of these two addressing types, various descriptions of different topics exist [6, 19, 18]. The important characteristics of these two addressing schemes are as follows.

- In *sender-initiated* multicast transport services, there are no special group addresses. When creating a multicast connection, a sender simply opens a connection and specifies a list of addresses which shall be associated with this multicast connection. This results in a star-shaped topology and consequently in a point-to-multipoint connection. Examples for sender-initiated multicast transport services are ATM and XTP.
- When using *receiver-initiated* addressing, special multicast addresses are used for sending data to a group. A multicast connection is opened by the receiver issuing a request to join a group, ie to receive the data that is sent to that group. The transport service then is responsible for arranging the routing appropriately. Because everyone can send data to the special group addresses, the topology is a mesh and the resulting connection type is multipoint-to-multipoint. An example for a receiver-initiated multicast transport service is IP multicast.

Obviously, receiver-initiated addressing is more flexible than sender-initiated multicast, because it makes multipoint-to-multipoint connections possible. On the other hand, there are several issues like reliability and ordering which favor sender-initiated approaches. However, a collaborative application should not be responsible for dealing with the addressing scheme used by a certain multicast transport service. Therefore, a support platform for collaborative applications should hide the addressing scheme from the applications. It must therefore have a way of mapping names onto both kinds of addresses[2], of establishing multicast connections, and it must also have a way to implement multipoint-to-multipoint connections on multicast transport services with sender-initiated addressing.

3.3 Data transfer

The ultimate goal of naming and addressing as described in the last two sections is to establish a connection between different entities using a collaborative application which can then be used to transfer data between them. Therefore, data transfer is the third area which is important for the support of collaborative applications. The requirements with respect to data transfer can be summarized as follows. It must be possible to transfer data to a group or to individual group members and it must be possible to receive data sent to a group or sent to the receiving entity individually. These requirements are motivated by the observation that in every collaborative application there is data to be transferred to all group members (such as updates of objects in shared whiteboards), and that in most applications it is also necessary to transfer data between individual group members (such as the request to unlock a specific object which is locked by one member of a group).

Because the support platform also needs to transfer data between different entities, control connections are required which can be used for that purpose. Control connections must be usable in the same way as the normal data transfer facilities, ie to all members of a group and to individual members. For example, if a new group member joins a group, it may be necessary (provided the multicast transport service uses a sender-initiated addressing scheme) for all other group members to join this user to their multicast transport connections. However, this should be achieved transparently for the collaborative application, which should only be notified of the fact that a new member has joined the group. The support platform would be responsible for adding this new member to the transport connections of all members. This would be most easily achieved if the joining member's support platform would send its address over the control connection to all members of the group.

An example for the use of control connections to individual members would be a group with restricted membership. Before joining this group, a user would have to consult one or more managers who are responsible for

[2] This mapping can not be a 1:1 mapping because in sender-initiated multicast transport services, group addresses do not even exist. Groups are addressed as lists of group members.

granting access to the group. These managers could be consulted using the control connections to the individual members. Only after the managers gave the permission to join the group, the new member would be admitted to it.

4 An architecture for the support of collaborative applications

In the previous sections it has been shown that communication support for collaborative applications is a task which may be divided into the three main areas name management, addressing, and data transfer. To reflect this modularity, the model presented here is divided into two major components (one dealing with names and addresses, the other one with connection setup and actual data transfer) which are shown in figure 1.

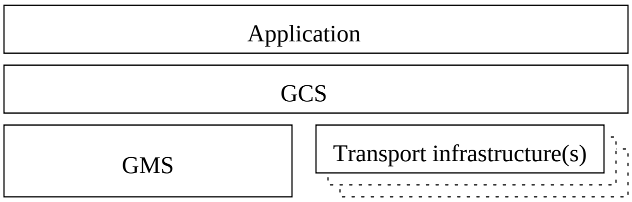

Figure 1: The architecture of the GMS/GCS

The *group management system (GMS)* is responsible for maintaining the name space and for the mapping of names to addresses. Because for some applications (such as name and group browsers[3], or asynchronous communications in mail based systems) it is sufficient to have naming and addressing capabilities, the GMS can be used on its own. It therefore has an interface which may be used from application programs. However, figure 1 depicts the case where both components are used.

The other component of the model is responsible for connection setup and actual data transfer and is therefore called *group communication support (GCS).* Because this component uses the addresses provided by the GMS, it cannot be used without it. The GCS component also is located on top of a transport infrastructure[4], which is used to accomplish the actual transfer of data between the distributed components of a collaborative application. The transport infrastructure is also used to transmit data that is important for the function of GCS.

The following sections describe the components of the GMS/GCS architecture in detail. Sections 4.1 and 4.2 give a definition of the two basic data structures of the GMS, which are users and groups. These two sections use an extended Backus-Naur formalism (EBNF)[5] notation to define the syntax of some attributes[6]. Table 1 gives a list of identifiers used in these definitions.

uid	is used to identify a user
gid	is used to identify a group
app	an identifier for any application which uses the GMS/GCS
UTPid	a unicast transport service identifier
uaddr	an address for a unicast transport service
RMTPid	a receiver-initiated multicast transport service identifier
SMTPid	a sender-initiated multicast transport service identifier
maddr	an address for a multicast transport service
ubid	identifies a user binding (described in section 4.1 and section 4.4)
gbid	identifies a group binding (described in section 4.2 and section 4.4)

Table 1: List of identifiers used for user and group attribute definitions

[3]A short description of how such a browser may be implemented on top of the GMS is given in section 5.1.

[4]Actually, it may also use different transport infrastructures at one time to be able to communicate with platforms which support a disjoint set of transport infrastructures.

[5]In EBNF, concatenation means that both parts have to be present. {} denotes the appearance of any number of the surrounded parts (including none). () is just for grouping purposes. | denotes the alternative of both parts, ie exactly one of the parts can be present. [] denotes optional parts which may be present or not.

[6]Only the attributes which are relevant for the structure of the entity space of users and groups are formally defined. The definitions of the other attributes (such as administrative information) are not given, because they are of less interest when looking at the model of users and groups.

In section 4.3, the GMS and the services it provides to application programs are described. These are services which are used to query and modify the name space, ie the set of users and groups. Finally, section 4.4 will show the services provided by the GCS component. With these services it is possible to set up connections to users and groups and to communicate with them.

4.1 Users

Users in the GMS are used to represent persons who use collaborative applications. Therefore it is necessary for each person using the GMS/GCS to be registered as a user. In the rest of the paper, it is assumed that this is the case, ie that each user in the real world is represented by a user object in the GMS. Users have a number of properties or attributes, which are described in the following list.

- *name* : uid

  Every user is identified by a unique name. This name is either used within the GMS (when a user is member, creator, or manager of a group) or from applications using the GMS. In particular, the name of a user is not depending on his location, ie it is location-transparent.

- *administrative information*

  The *administrative information* for each user contains data like the postal address, maybe an office number, the dates and times of the last bindings, and the electronic mail address. Because asynchronous communication sometimes is the only option (eg when a user is not active at the time of a synchronous activity), every user should also be reachable via electronic mail. The e-mail address registered within this attribute can therefore be used to contact users which are not bound to GCS (see section 4.4 for an explanation of GCS binding).

- *access rights*

  The *access rights* of a user specify who is authorized to request information about that user. The access may be open, restricted to users who are members of a group this user is a member of, or to active users of one of these groups.

- *identification information*

  When accessing the data stored in the GMS (either reading or modifying something), it is necessary to identify oneself. Only after performing an identity check based on the *identification information*, the system grants access to its data. Consequently, when binding to the GMS (see section 4.3 for an explanation of GMS binding), this information is used to authenticate a user.

- *bindings* : { ubid app UTPid uaddr (RMTPid | (SMTPid maddr)) [gbid]}

  Each user can have zero, one, ore more bindings. A user with no bindings is said to be unbound. Being unbound means that this user currently is not actively working with any application which uses GCS. Each binding consists of information about applications and transport services. The first value identifies the application through which this user is bound. The next two values are a pair of a unicast transport service identifier and a unicast address. Two more values are a pair of a multicast transport service identifier and possibly an address for this service (the address is only necessary in case of sender-initiated multicast transport services). The last value identifies the group for which this binding is used (which is the case after the user has bound to a specific group, see section 4.4 for an explanation of GCS group bindings).

- *groups* : { gid }

  This attribute is a list of groups the user ever joined (and did not subsequently leave). Therefore, it is not a list of all groups the user is a member of, because it lists only direct memberships. Indirect memberships (which may be achieved by a group he is member of becoming a member of some other group) cannot be determined by interpreting this attribute only.

- *managing* : { gid }

  Any user maybe manager of one ore more groups. For every group a user is managing, there is an entry within this attribute. Depending on the access rights of a group, different tasks have to be carried out by a group's manager. The manager of a group is not necessarily a member of this group.

These attributes are used for the description of users. How these attributes are used by the GMS is described in section 4.3, and how they are used by GCS is described in section 4.4. The next section will give a definition of the attributes which are used to describe groups.

4.2 Groups

A central notion of the GMS/GCS architecture is that of groups. Groups may be used to identify sets of members which can be addressed in a collaborative application. Groups are used to represent groups of collaborating users, which are either direct or indirect members of the group. The following list describes all attributes that are defined for groups.

- *name* : gid

  The name of a group is used to refer to a group. References to a group are possible from within the GMS (when a group is member, super-group or manager of another group) or from applications using the GMS.

- *administrative information*

  Administrative information is used to keep data which is not important to the structure of a group. This includes creation, modification, and access dates, the group's creator, a textual description of the group, and some other information which may be interesting for administrators.

- *access rights*

  The *access rights* of a group are used for a number of purposes. They specify whether a group's *managers* (one or all of them) must be consulted when joining or leaving a group, or whether it may be joined or left without such restrictions. In the first case it is impossible to join or leave such a group when no manager is currently bound to GCS. The *access rights* are also used to determine who may read the *administrative information* of a group and who is allowed to look at its *bindings*. It is also specified whether *managers* and/or *members* are notified of any changes of the group's *members*. However, only active members are notified.

- *maximum number of members*

  Because groups are not necessarily unlimited in size, this attribute may be used to limit the maximum number of members. Notice, however, that it is only possible to effectively limit the number of users if *subgrouping* is not allowed (because this attribute only limits the number of direct members). Otherwise, any group which is a member of a given group may have more members than specified (if they are not limited themselves).

- *subgrouping*

  This flag indicates whether groups may be used as members of a group. If it is set to disallow subgrouping, only users are allowed as members, thus making it possible to effectively limit the number of users in a group (see also attribute *maximum number of members*). This attribute also needs to be set to disallow subgrouping when a group shall be *managing* another group.

- *bindings* : { gbid app ((RMTPid maddr) | (SMTPid { maddr }))}

  A group may be bound to specific applications with specific transport services. Every binding is characterized by the application this binding is to be used for, the used transport service, and addressing information. Depending on how multicast addressing is done in the selected transport service (see section 3.2 for a description on different multicast addressing schemes), the addressing information may be a single address which identifies the group (in case of receiver-initiated addressing), or a set of addresses which identify all currently active group members (in case of sender-initiated addressing)[7].

- *members* : {(uid { ubid }) | (gid { ubid { gid }})}

  The members of a group are defined by a set of users and groups. If *subgrouping* is set to no, then only users are allowed to be members. Furthermore, the number of elements in the member set is limited to the *maximum number of members*, if this attribute is set. Each member is either active or passive. Being active means that this particular member is bound to this particular group (using the GCS *bind user* and *bind to*

[7]If the addressing information is a set of addresses which identify all currently active group members, then it is updated with every user changing his status from passive to active or vice versa

group services described in section 4.4). In case of an active group, it is also registered which members of this group are active.

- *managers* : { uid | gid }

  This is a list of user and/or groups who are responsible for managing the group. Based on the *access rights*, particular actions involving a group (such as joining it, or making it a member of another group) needs to be confirmed by the *managers*. Depending on the *access rights*, only one or all *managers* have to give their confirmation. Groups are only allowed to be *managers* if they do not allow *subgrouping*.

- *groups* : { gid {(uid | gid) gbid { gid }}}

  This attribute lists all groups of which the group to be described is a direct member. Indirect memberships can not be determined by interpreting this attribute alone. The *groups* attribute also lists for all super-groups of the current group, whether there are active members who are bound to GCS, and which groups are involved in this relationship between the current group and this particular super-group (for a detailed discussion of this issue see the description of the *bind to group* service in section 4.4).

- *managing* : { gid }

  A group maybe manager of one ore more groups. This means that all *members* of that group are managing the group. Only groups with no *subgrouping* are allowed to be managers of groups. For every group a group is managing, there is an entry within this attribute. Depending on the access rights of a group, different tasks have to be carried out by group managers. The manager of a group is not necessarily a member of this group.

These attributes are used to define a group. Together with users, these are the only entities used in the name space of the GMS. However, because of these definitions, a lot of relations between users and groups, or groups and groups, are possible. Using these relations (which can be created and modified using the services described in the next two sections) it is possible to reflect the relationships between collaborative workers and groups of collaborative workers who are using a collaborative application based on GMS/GCS.

4.3 GMS services

Group management system (GMS) and group communication support (GCS) are two separate entities. GMS services are used by the GCS layer. They can also be used by applications which do not need communication support but access to the database which consists of the entities described in the last two sections, ie users and groups.

Because GCS needs to modify some information that shall not be modified by any other application, two interfaces exist, one being public and one being exclusively for GCS. In the following list, only the public services are described. The GCS interface to the GMS is not accessible for applications and therefore only the access through GCS services is described in the subsequent section, which describes GCS services.

- *create*

  This operation may be used for creating users or groups. When creating a user, all information that is necessary for the description of a user (as described in section 4.1) has to be supplied. When a group is created, it is necessary to be bound to the GMS because the identity of the creator must be known to the system.

- *bind*

  Binding to the GMS requires a name and the identification information that is necessary to identify oneself as this specific user. Only after binding to the GMS it is possible to request information about users and groups and to change any information (ie join or leave groups or modify entities). Binding to the GMS is not the same as binding to GCS (using the GCS *bind user* service), which is described in section 4.4.

- *query*

  Sections 4.1 and 4.2 have shown that there is much more information available than just user and group names. This information can be accessed by using the *query* service. When querying information for a certain entity, the identity of the querying user and the *access rights* of the entity being queried are checked. If the user is authorized to query some or all information about the requested entity, then this information is returned. Otherwise, the *query* service fails with a notification about insufficient authorization.

- *modify*

  Attributes of entities can also be modified. However, this is only true for non-structural attributes[8] (structural attributes are implicitly modified by other GMS and GCS services). Provided a user is sufficiently authorized, non-structural attributes of entities may be modified. However, a consistency check is performed to make sure that such a change does not create an inconsistency of the entity space, eg setting the *maximum number of members* of a group lower than the actual number of members or disallowing *subgrouping* in a group which has groups as *members*.

- *join*

  The *join* service is used to join a given user or group to a group. Provided the user performing the join is sufficiently authorized, and the *maximum number of members* is not reached yet (if it is set), the join can be performed. It may be necessary to consult the *managers* of a group about a join request. In case of a group being joined to another group, it is also checked that *subgrouping* is not disallowed in the group which shall become the super-group of the group being joined. It is also checked for recursions, ie it is guaranteed that the graph of all group member relationships always is an acyclic graph. Joining means that the *name* of the entity being joined is added to the *members* of the super-group and that the *name* of the super-group is added to the em groups of the entity being joined. Depending on the *access rights*, the active *members* and/or *managers* of the group being joined are notified about the join.

- *leave*

  Leaving a group is the service which removes particular *members* from a group. It may be necessary to consult the *managers* of a group about a leave request. Only passive members may leave a group, thus active members first have to use the *unbind from group* service described in section 4.4. If a group is leaving a group, then the leaving group must be passive in the group being left, ie there are no bindings which depend on the relationship between these two groups. Before doing anything else, it is first checked whether the requesting user is sufficiently authorized to perform the leave. This check is based on the *access rights* of the group to be left. Both users and groups may leave a group. It is checked whether the leaving entity is a direct *member* (ie listed in the *members*) of the group it wishes to leave. If so, the group's name is removed from the entity's *groups* and the entity's name is removed from the group's *members*. Depending on the *access rights*, the active *members* and/or *managers* of the group being left are notified about the leave.

- *delete*

  Entities may also be deleted, ie removed from the GMS name space. In the first place, it is checked whether the user requesting the delete service is sufficiently authorized. In case of a user being deleted, no group bindings are allowed to exist. In case of a group being deleted, all *members* of the group must be passive (ie not being bound to that group or any of its *groups*), and there are only users left as *members* (ie there are no subgroups). Deleting a user removes his name from all group *members* lists of his *groups* and from all groups' *managers* lists he is *managing*. Deleting a group removes its name from the *members* lists of all its *groups*, from the *groups* lists of all its *members*, and from the *managing* lists of all its *managers*. Depending on the *access rights* of the group being deleted, the active *managers* are notified of the deletion.

- *unbind*

  Unbinding from the GMS is the service which ends the interaction with the GMS. After the *unbind* service has been performed, the identity of a user is not longer known. Therefore, none of the querying and modification services described above which need any kind of authorization may be performed.

In addition to these public services, there is a set of services which are only accessible for GCS. For example, GCS needs a way to modify bindings of users and groups when they bind to or unbind from GCS (using services described in the next section). However, because these services are not accessible for applications using the GMS/GCS, they are not described in this paper.

[8] Non-structural attributes are attributes whose value is not related to the relationships between entities or between entities and bindings. All attributes of sections 4.1 and 4.2 without an EBNF definition (such as *access rights* and *maximum number of members*) are non-structural attributes.

4.4 GCS services

In addition to the GMS services described in the last section, there is a number of services GCS is providing. According to the GMS/GCS architecture shown in figure 1, the GCS component uses both the GMS and one or more transport services. Consequently, all services described in the following list are using these two resources. When using GCS, it can be in one of three state types. Figure 2 shows the state types and the services which can cause the transitions between the different state types.

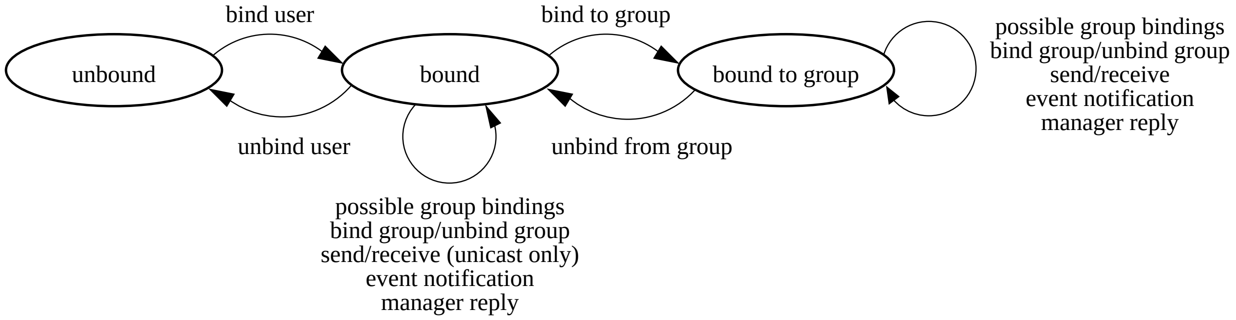

Figure 2: State diagram for GCS operations

The states "bound" and "bound to group" actually are state types, because an application may be bound to different transport services, using each one for another communication. It may also be bound to a number of groups. This may happen when different transport infrastructures are used or when a user needs to be active in more than one group. In this case, there are states "bound to group 1", "bound to group 2", and "bound to groups 1 and 2" which may be entered or left by performing the *bind to group* or *unbind from group* services with the appropriate group bindings.

The following list describes all services provided by GCS. It also describes what happens when these services are invoked by an application in terms of attribute accesses of users and groups involved.

- *bind user*

  Binding to GCS is necessary to provide information about the application being used and the available transport services. Consequently, an application has to specify the application type, and unicast and multicast transport services. After making sure that the transport services requested are available, a *binding* for the user is created (as described in section 4.1) and transport connections are established. After being bound, the user's binding is not associated with a group (the *bind to group* service has to be used to establish a group binding), consequently he can only *send* and/or *receive* unicast data. It is possible to have more than one binding, eg when using different transport infrastructures for different groups.

- *bind group*

  A group binding can be created using the *bind group* service. Using this service, it is necessary to specify an application which is associated with the binding, and a multicast transport identifier. If the multicast transport service being used is receiver-initiated, then a group address must also be specified. If the multicast transport service is sender-initiated, the list of addresses initially is empty and grows with every user binding to the group with that group binding. Creating a group binding is only possible for sufficiently authorized users, depending on the *access rights* of the group.

- *possible group bindings*

  After binding to GCS, it can be determined which group bindings are possible (based on a given user binding). This can be evaluated by querying the appropriate information from the GMS. Using the given binding and the *groups* a user is member of, it is easy to also find all indirect memberships (the *groups* of the *groups*, and so on) and to find all possible group bindings. A group binding can be used by a bound user if his binding matches both the application and the transport service of one of a group's *bindings*. All group bindings that may be established are returned by the *possible group bindings* service.

- *send*

  Data can be sent to users or groups. In both cases, a binding must be specified which identifies the connection to be used. Data can only be sent to users who are members of the same group as the sending user, and to groups in which the sending user is active.

- *receive*

  Receiving data can be done either synchronously or asynchronously. Synchronous receiving waits until some data is arriving, while asynchronous receiving results in a software interrupt when data arrives. To handle asynchronous receiving, a user has to install a procedure which handles these asynchronous events. The data being received may have been sent to a group the receiving user currently is active in or to the user individually. Therefore, data always is marked with the binding to which it has been sent (which may be a user's or a group's binding), and with the binding from which it has been sent (which always is a user's binding).

- *event notification*

  A number of asynchronous events can occur when a user is bound to GCS. These events include managing events (which may occur when a new user wants to *join* or *leave* a group which appropriately set *access rights*), events for active group members (which may occur when a user binds to or unbinds from the group) and events which inform of certain conditions that may occur because of other users (such as a group of which the user is a member and which is removed by a manager). Because an *event notification* always is asynchronous, a user has to install event handlers which are activated upon the occurrence of the corresponding event. Events which require management actions (such as allowing or denying access to a group) can be handled using the *manager reply* service to give the appropriate reply.

- *manager reply*

  After a manager of a group has received an *event notification* indicating an event which requires management actions, the *manager reply* service has to be used to specify the action to be taken. A management action is the reply to a *join*, *leave*, or *delete* request which involves the group which is managed by the user receiving the *event notification*. Depending on the *access rights*, one or all managers must confirm such a request. Only after conformation by one or all managers, the requested service (*join*, *leave*, or *delete*) is executed for the requesting user. If no manager is currently active or if none of the active managers is responding, the requested service fails indicating the reason.

- *bind to group*

  To become an active group member of a group, a user has to use the *bind to group* service. This service can only be used after the *bind user* service has successfully established a user binding. In order to bind to a group, a user has to specify the group binding and the user binding to be used. The *possible group bindings* service described earlier gives a list of all group bindings which a user may use given an existing user binding. It is first checked whether the group binding is possible, ie the user has to be a member of the group and the user's and the group's binding have to match with respect to the application and multicast transport identifiers. After these checks the binding can be established. Establishing a group binding is different depending on the type of the multicast transport type.

  - In case of a receiver-initiated multicast transport type, the receiver (ie the user being bound) has to arrange for joining the multicast group which corresponds to the group binding. This is accomplished by invoking the appropriate multicast transport service. Furthermore, the user's binding is updated with the multicast address and the group binding to which this binding corresponds.
  - Sender-initiated multicast transport requires a different procedure. In this case, a multicast connection to all other active group members has to be established and they all have to add the newly bound user to their multicast connections. This is done by the following procedure. In the first place, the list of all active group members' addresses is taken from the group binding and a multicast connection to these members is established. Then the address of the new member is transmitted to the GCS components of all active members (using the multicast control connection described in section 3.3). Now each of them can add this new member to their multicast connections. The new member's address is also added to the group binding's multicast addresses. Finally, the user's binding is updated with the group binding to which this binding corresponds.

  After these actions, the connections are set up and the user's binding is updated correctly. However, the user still needs to be marked as active. Depending on whether he is a direct or indirect member of the group to which he has bound, this is also done with different procedures.

  - If the user is a direct member of the group to which he has bound (ie he is listed in its *members*), then the user binding is added to the user in the *members* list, marking this member as being active.

  - If the user is an indirect member of the group to which he has bound (ie he is not listed in its *members* but – through one or more levels – in the *members* of one of its *members*), then the binding has to be registered differently. At first, the user binding is added to the group within the *members* list through which this user is an indirect member. Also added is a list of groups which establish the indirect membership relationship. This process is continued for every group until the group of which the user is a direct member of is found. This way, all memberships through which the user is an indirect member of the group to which he has bound are marked as active. Furthermore, all the *groups* lists of the groups beneath the group to which the user has bound, are updated in a similar way (using the group binding and the names of the super-groups), always indicating to which group the binding belongs.

  Obviously, *bind to group* is a complex service and requires the modification of a number of attributes of different entities. This is necessary because the activity of the user who is bound to a group has to be reflected in all involved entities. Only then it is possible to deny conflicting operations which would create inconsistencies, such as removing a group which constitutes the indirect membership of the user in the group he is bound to.

- *unbind from group*

  Unbinding a user from a group binding can be done by using the *unbind from group* service. Unbinding from a group with a given binding means that a user changes his state from active to passive, ie he is not longer reachable as an active member of the group. It is first checked which kind of multicast transport service is used for the group from which the user wants to unbind. Depending on the type, different actions are necessary.

  - In case of a receiver-initiated multicast transport type, it is only necessary to notify the transport service of the fact that the address registered in the group binding is not longer needed.
  - Sender-initiated multicast requires another procedure. Because all other active group members have connections to the user who wants to unbind from the group, these connections have to be closed. This is done by notifying all group members of the unbind of this user, using the GCS control connection. After that, the multicast connection of the unbinding user is closed. This way, all communication connections are closed. Now the address of the unbinding user is removed from the group binding's list of multicast addresses.

  After these actions, the group binding is removed from the user's binding. The last modification that is necessary is to remove the user binding from the *members* list of the group which marks this user as being active with this user binding. If the user is a direct member of the group, the user binding which identified the group binding is removed from the member name in the *members* list. If the user is an indirect member, all the settings described in the *bind to group* service needs to be taken back. This finally also results in the user no longer being marked as active with this particular binding.

- *unbind group*

  Deleting a group binding is possible using the *unbind group* service, provided, the user is sufficiently authorized (depending on the *access rights* of the group). A group binding can only be deleted when no user is currently bound to the group with this particular binding. Deleting a group binding simply removes this binding from the *bindings* of the group.

- *unbind user*

  Unbinding from GCS removes a user binding. First it is checked whether the binding to be unbound is bound to a group. If it is, the user first has to unbind from that group. Otherwise, all transport connections associated with the user's binding are closed and the binding is removed from the user's list of *bindings*.

These are the services provided by the GCS. The next section will give an example how these services can be used. However, it has to be kept in mind that these services are not intended to be directly presented to a user, but to be used by application which can use different representations at the user interface.

5 Applications

Given the architecture described in the last section, it is now possible to describe applications using the GMS/GCS. The following two sections will give descriptions of applications. While the first application is a rather simple name space browser which only uses the GMS services, the second application described is a shared white-board, one of the most popular collaborative applications in the research field. However, we will focus on which services of the GMS/GCS are being used instead of the functionality of the white-board.

5.1 Name space browser

In section 4 it has been described that the GMS may be used without GCS for some applications. One such application is a name space browser which makes it possible to create a visual representation of the name space provided by the GMS. This browser would get the information about different entities using the *query* service of the GMS. For example, a group could be displayed as a circle with smaller circles representing the *members* of the group. Selecting a member would then *query* the information of that entity. Another possible representation would be a tree-based display which would display the hierarchical relationships between entities. A user of the name space browser would then be able to explore the part of the name space he is interested in, perhaps using the *modify* service changing some attributes. However, to *modify* entities or even to *query* entities which are not open to the public (ie have their *access rights* set to restrict access to certain attributes), it would first be necessary to *bind* to the GMS.

5.2 Shared white-board

The shared white-board is an application which is built on top of GMS/GCS, ie it uses the services described in sections 4.3 and 4.4. A first activity of a user of this application would be to *bind* to the system, ie to identify himself. Provided the user identification was successful, it would then be necessary to create a user binding, ie to use the *bind user* service. There it would be necessary to specify the application type (ie shared white-board) and the available transport services, eg IP unicast and multicast. Now it would be possible to use the *possible group bindings* service to get a list of all groups the user could be bound to. This list would be a list of all groups the user is direct or indirect member of and which have a group binding that matches the user's binding (ie has the same application type and multicast transport service). Using this list, it would be possible to give the user of the shared white-board a list of groups he may bind to. It would also be possible to *query* the GMS for some or all groups in order to find out who is currently bound to these groups, and thus to give a more detailed description of the groups to which the user may be bound.

After selecting one of the groups, the user could join the collaborative white-board session by using the *bind to group* service. Depending on the design of the white-board, it could first be necessary to use a unicast connection to one of the active group members to get the current state of the white-board. After this update, the application would use all data being received for that group binding to draw the shared white-board. Provided the user interface offers these capabilities, it would also be possible to use unicast connections to group members to exchange private data (such as short messages). Another function could be achieved by dynamically creating subgroups, eg for painting sketches in smaller groups, and bind to them for the duration of these activities. Generally, using the *create*, *join*, *leave*, and *delete* services of the GMS and the *bind to group* and *unbind from group* services of GCS it would be possible to perform flexible configurations among the participants of the collaborative white-board session. It would also always be possible to identify the data being received, because the *receive* service always identifies to which binding the data being received was sent. Finally, the collaborative session would be finished using the *unbind user* and *unbind* services.

6 Further work

The design of the GMS/GCS architecture currently concentrates on the aspects of naming/addressing and the setup of connections. However, it would be possible to look at some other areas in greater detail. This section will describe topics which are of further interest in the area of communication support for collaborative applications.

Up to now, the object types of the GMS (ie users and groups) are defined by attributes which are required for the operation of the GMS/GCS. However, application dependent data storage within these objects could be very useful. For example, it would be possible to define an additional attribute *application data* which could be used by applications to store application dependent data (together with the application identifier) which could then be accessed every time the application uses this entity. This could eliminate the need for applications to store configuration data somewhere else, where it is not accessible for another instance of that application invoked at another place. However, the locking of this data (assumed two instances are running concurrently), and the amount and modification frequency are important issues when such an attribute is invented.

Another important issue are quality of service (QoS) aspects, which are also sometimes called application requirements. Currently, these requirements are not addressed in the GMS/GCS architecture, but work on this topic is being done and will be integrated in the future. However, it will be extremely difficult to deal with QoS specifications when a heterogeneous transport infrastructure is present. Because existing multicast services such as XTP, ATM, or IP multicast have different QoS architectures (or none at all), one single QoS framework is hard to define. In the context of GMS/GCS, two different questions need to be discussed.

- The application using GCS may want to specify a set of quality parameters for data transmission. These parameters specify the reliability and the characteristics of data transmission (such as bandwidth, delay, delay jitter, or the error rate). For a discussion of these parameters see the article by Heinrichs et al. [10]. Based on the multicast transport service being used, these requirements can or can not be satisfied. It is then up to the application to choose another transport service (ie to establish another user binding), or to adapt its requirements to the QoS being offered.

- The other aspect of QoS is the interface that is available on given transport infrastructures. This interface may support QoS specifications or it may just offer a best-effort approach (such as IP multicast). Based on this interface, the QoS requested by the application may or may not be satisfied. Because GCS also uses multicast over control connections (such as the dissemination of a new active group member within the *bind to group* service), it may become questionable whether a non-reliable multicast transport service should be used for this purpose. Alternatively, it could be decided by GCS to use the multicast service only when it is *all-reliable* (as defined by Cheriton [4]) and to use the unicast control connections otherwise.

Another interesting field is that of additional modules which might be added to GCS, sitting on top of it and offering additional services to applications. One such example would be a “multicast gateway”, the design of which is given in figure 3. The task of such a gateway would be to make possible the interworking of two groups using different transport infrastructures. This could be done by forwarding data from transport infrastructure A to transport infrastructure B and vice versa. However, this task can become difficult when the characteristics of the two infrastructures are different.

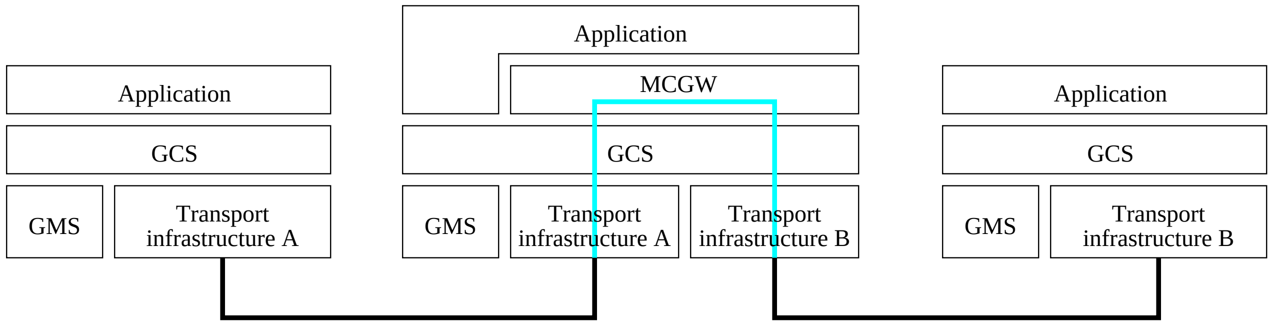

Figure 3: A gateway for different multicast transport services

Carle et al. [3] describe a hardware group communication server which could also be used as a “multicast gateway”. Another possible approach for the support of heterogeneous multicast would therefore be the usage of such a specialized server. However, the location, management, and reservation of these servers would be a complicated task, and the resulting design would be completely different from the one shown in figure 3.

7 Conclusions

In this paper, the GMS/GCS architecture has been presented which serves as a communication support platform for collaborative applications. The three main advantages when using this platform are the reduction of implementation efforts, the sharing of user and group descriptions among different collaborative applications, and the communication abstraction which makes the application independent from the underlying transport service.

The architecture consists of a *group management system (GMS)* and a component for *group communication support (GCS)*. The GMS is responsible for managing the name space of the GMS/GCS, which is constructed of users and groups. GCS uses the services of the GMS and one or more transport services to provide applications with communication support. The objects of the GMS, ie users and groups, and the services provided by the GMS and GCS have been described in detail. Two application scenarios have been described and some further work has been outlined.

References

[1] ATM Forum. *ATM User-Network Interface Specification Version 3.0.* Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[2] Thomas Bannon and Ivor Page. group: A Distributed Group Specification and Management Service. In *UNIX – The Legend Evolves. Proceedings of the Summer 1990 UKUUG Conference*, pages 61–76, Buntingford, UK, 1990. UKUUG.

[3] Georg Carle, Jochen Schiller, and Claudia Schmidt. Support for High-Performance Multipoint Multimedia Services. In D. Hutchison, A. Danthine, H. Leopold, and G. Coulson, editors, *Multimedia Transport and Teleservices – Proceedings of the International COST 237 Workshop*, volume 882 of *Lecture Notes in Computer Science*, pages 219–240, Vienna, November 1994. Springer-Verlag.

[4] David R. Cheriton. Request-Response and Multicast Interprocess Communication in the V Kernel. In Günter Müller and Robert P. Blanc, editors, *Networking in Open Systems*, volume 248 of *Lecture Notes in Computer Science*, pages 296–312, Oberlech, Austria, August 1986. Springer-Verlag.

[5] Stephen Edward Deering. *Multicast Routing in a Datagram Internetwork.* PhD thesis, Stanford University, 1991.

[6] Luca Delgrossi, Ralf Guido Herrtwich, Carsten Vogt, and Lars C. Wolf. Reservation Protocols for Internetworks: A Comparison of ST-II and RSVP. In D. Shepherd, G. Blair, G. Coulson, N. Davies, and F. Garcia, editors, *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, volume 846 of *Lecture Notes in Computer Science*, pages 195–203, Lancaster, UK, November 1993. Springer-Verlag.

[7] Flavio Depaoli and Francesco Tisato. Coordinator: A Basic Building Block for Multimedia Conferencing Systems. In *Proceedings of the GLOBECOM'91 Conference*, pages 2049–2053, Phoenix, Arizona, 1991. IEEE Computer Society Press.

[8] Flavio DePaoli and Francesco Tisato. A Model for Real-Time Co-operation. In Liam Bannon, Mike Robinson, and Kjeld Schmidt, editors, *Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, pages 203–217, Amsterdam, September 1991. Kluwer Academic Publishers.

[9] Geert J. Heijenk, Xinli Hou, and Ignas G. Niemegeers. Communication Systems Supporting Multimedia Multi-user Applications. *IEEE Network*, 8(1):34–44, 1994.

[10] Bernd Heinrichs, Kai Jakobs, and Alessandro Carone. High performance transfer services to support multimedia group communications. *Computer Communications*, 16(9):539–547, 1993.

[11] Larry Hughes. A Multicast Interface for UNIX 4.3. *Software – Practice & Experience*, 18(3):15–27, 1988.

[12] ITU. The Directory – Overview of Concepts, Models and Services. Recommendation X.500, March 1995.

[13] S. Kille. A String Representation of Distinguished Names. Internet RFC 1779, March 1995.

[14] S. Kille. Using the OSI Directory to Achieve User Friendly Naming. Internet RFC 1781, March 1995.

[15] T. Kirsche, R. Lenz, H. Lührsen, K. Meyer-Wegener, H. Wedekind, M. Bever, U. Schäffer, and C. Schottmüller. Communication support for cooperative work. *Computer Communications*, 16(9):594–602, 1993.

[16] Rob Kling. Cooperation, Coordination and Control in Computer-Supported Work. *Communications of the ACM*, 34(12):83–88, 1991.

[17] Andreas Mauthe, David Hutchison, Geoff Coulson, and Silvester Namuye. From Requirements to Services: Group Communication Support for Distributed Multimedia Systems. In Ralf Steinmetz, editor, *Proceedings of the Second International Workshop on Advanced Teleservices and High-Speed Communication Architectures*, volume 868 of *Lecture Notes in Computer Science*, pages 266–279, Heidelberg, Germany, September 1994. Springer-Verlag.

[18] Danny J. Mitzel, Deborah Estrin, Scott Shenker, and Lixia Zhang. An Architectural Comparison of ST-II and RSVP. In *Networking for Global Communications – Proceedings of the IEEE INFOCOM '94 Conference on Computer Communications*, pages 716–725, Toronto, 1994. IEEE Computer Society Press.

[19] Sridhar Pingali, Don Towsley, and James F. Kurose. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. In *Proceedings of the ACM Sigmetrics Conference on Measurement & Modeling of Computer Systems*, pages 221–230, Nashville, Tennessee, May 1994.

[20] T. Rodden, J. A. Mariani, and G. Blair. Supporting Cooperative Applications. *Computer Supported Cooperative Work*, 1(1–2):41–67, 1992.

[21] Tom Rodden and Gordon S. Blair. Distributed systems support for computer supported cooperative work. *Computer Communications*, 15(8):527–538, 1992.

[22] Luís Rodrigues and Paulo Veríssimo. xAMp: a Multi-primitive Group Communications Service. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pages 112–121, Houston, Texas, 1992. IEEE Computer Society Press.

[23] Mark Roseman and Saul Greenberg. GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications. In John Turner and Robert Kraut, editors, *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 43–50, Toronto, November 1992. ACM.

[24] J. Saltzer. On the Naming and Binding of Network Destinations. Internet RFC 1498, August 1993.

[25] John F. Shoch. Inter-Network Naming, Addressing, and Routing. In *Proceedings of the Seventeenth IEEE Conference on Computer Communication Networks*, pages 72–79, Washington, D.C., 1978.

[26] W. Timothy Strayer, Bert J. Dempsey, and Alfred C. Weaver. *XTP: The Xpress Transfer Protocol.* Addison-Wesley, Reading, Massachusetts, 1992.

[27] Paulo Veríssimo and Luís Rodrigues. Group Orientation: A Paradigm for Distributed Systems of the Nineties. In *Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems*, pages 57–63, Taipei, Taiwan, 1992. IEEE Computer Society Press.

[28] Paulo Veríssimo and Werner Vogels. The Changing Face of Technology in Distributed Systems. In *Proceedings of the Fourth Workshop on Future Trends of Distributed Computing Systems*, pages 119–127, Lisbon, 1993. IEEE Computer Society Press.

[29] Alfred C. Weaver. The Xpress Transfer Protocol. In *Proceedings of the International Workshop on Advanced Teleservices and High-Speed Communication Architectures*, pages 253–259, Munich, March 1992.