Modelling Groups for Group Communications

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

Abstract

This paper presents a general model of a Group Management Service (GMS) which is designed to support collaborative interactions among groups of distributed users using different applications. There are two main benefits of such a service. Firstly, it would be easier to implement new collaborative applications because of the possibility to use an existing service. Secondly, it would be possible for different applications to share collaboration relevant information because of a common database of information about users and groups maintained by the GMS. One important property of the GMS is its flexibility with respect to the information stored. It is possible to store application-independent as well as application-dependent information. Using an object-oriented approach, applications can share the application-independent information (such as a group's members and administrative information) and can also use the GMS to store application-dependent information which can only be interpreted by a closed set of applications (those who know the syntax and semantics of the application-dependent information). The model of the GMS is very simple and consists mainly of two classes of objects, namely user and group. A small set of operations is provided for querying and modifying GMS information. The possibility to store application-dependent information is realized by allowing using application to create derived classes (ie subclasses) of the classes user and group. Thus it is possible for applications using the GMS to implement their own user and group classes without losing the ability to manage these objects with the GMS. Two applications are presented which may use the GMS to manage their users and groups. Both applications use application-specific derived classes of user and group. However, it is still possible for these applications to share the application-independent information of their users and groups.

1 Introduction

The emerging need for software supporting group cooperation is indicating that computers are increasingly used for collaborative group work. Rodden, Mariani and Blair [13] point out that the "traditional" support offered by operating systems often fails to be useful for cooperative applications. A result of this situation is the development of groupware (as we will call software supporting collaborative work throughout this paper) today, where every application implements its own set of functions for supporting collaboration. This is not only highly inefficient in terms of development time, it also makes it nearly impossible for different groupware applications to share the same collaboration relevant information. This leads to the somehow absurd situation that collaborative applications are not able to collaborate among themselves.

Consequently, it is necessary to identify certain areas of support for groupware applications which are of interest to a large set of potential applications. This would not only simplify the development of new applications by providing them with sophisticated support. It would also allow the sharing of collaboration relevant information between different applications because of the common platform they are built on. This issue is very important in the development towards more open CSCW systems as described by Navarro, Prinz and Rodden [9].

One key issue for every groupware application is the management of groups, where groups are used to identify people with a common goal or with other common properties. A service offering support for this task in a flexible and general way is clearly needed to make the sharing of group related information possible. We call such a service a group management service (GMS). This service will implement the distributed management of users and groups and the information associated with them.

Veríssimo and Rodrigues [15] describe how this type of service is needed on a system level, assuming a network of distributed systems which need means to form groups of systems. Seeing that the usage of the computer also becomes more and more a distributed activity, this type of support is also necessary on a user level. The provision of such a service would make it much easier for groupware applications to manage users and groups without having to implement this functionality, to use the data of users and groups already provided by other applications, and to make themselves independent of the underlying infrastructure needed to implement such a distributed service.

The structure of the paper is as follows. Section 2 describes the properties required by a GMS in order to be useful for groupware applications. Section 3 gives a survey of some work which is related to the model described in this paper. Section 4 describes a model of a GMS which is suitable to fulfil the given requirements. Section 5 discusses some implementation issues and Section 6 gives two examples of applications and their way of applying the model described. Finally, section 7 points out which way we will go in the future, and section 8 gives the conclusions.

2 Requirements

While a GMS will be a useful component of a platform for groupware applications, it will only be used if it satisfies some requirements. This section will give a set of properties which are needed to construct a service which is useful for as many applications as possible. The following list gives a very abstract impression of what is required from a GMS.

- Simplicity

  Simplicity refers to the fact that the service should be easy to use. This means that both the model and the operations should be easy to understand and to apply to given problems. Intuitive ways of modelling users and groups have to be supported. The semantics of groups lie within the responsibility of the applications creating and/or using them.

- Flexibility

  Flexible support of modelling groups must be able to support any model of users and groups used by collaborative applications. Flexibility can be viewed in two ways. The first view is to provide a flexible way to handle users and groups, ie it must be easily possible to create, modify and delete users and groups and to change the membership of users and groups within groups. The second view of flexibility is the requirement to be able to store application-specific group related information. The GMS therefore must provide the possibility to handle application-specific groups, which can be used by both the application which created them and any other application using the GMS (which will only be able to access the application-independent properties of the group).

- Generality

  The model should be general enough to support a wide range of software designed for group usage. Existing concepts of users and groups should be subsets of the new model, in order to make sure that the new model is general enough to fulfil the requirements of groupware applications. The new model must be usable by different applications concurrently and it must also be possible to define additional attributes and semantics for users and groups in order to fit the model to application-specific needs.

- Scalability

  The model defined here should be scalable, ie there should be no limitations for applying the model to domains of any size. Scalability applies to the number and sizes of groups as well as to the distribution of group members. Group members should be able to be distributed globally and it should be possible to (possibly temporarily) create and maintain a huge number of groups without problems.

This list of requirements does not anticipate any specific model or architecture. It will be used to assess related work in the following section and to define a model for a GMS in section 4.

3 Related work

This section will discuss some work which is related to the GMS. So far only little work has been done to design and implement a general and application-independent group management service. Therefore, also work only partly related to the GMS has been taken into consideration.

A very simple implementation of users and groups can be found in the Unix operating system as described by Winsor [17]. Users are members of possibly several groups with one being the primary group. Groups are not hierarchical, ie groups can not be defined by other groups. Group membership is rather static, there is no programming interface for changing users or groups. Modifications are done by editing special files which can be distributed over a network using the Network Information Service NIS. Winsor [16] describes NIS+, which is the successor of NIS. NIS+ allows for better distribution of the users and groups databases, however, it does not change the simple model of Unix users and groups.

In an early paper about the support of group communications, Prinz and Speth [11] describe the AMIGO approach. AMIGO is a project which studied messaging in group environments. Because of its focus on message systems (X.400 in particular), its relation to the GMS is limited to specific aspects. However, several aspects of this approach, such as the attributes of groups and the operations related to groups and members are of interest for the GMS. The limitations of the message based approach and the inability to extend the set of attributes (described as flexibility in section 2) make it impossible to use this model as a general platform for groupware applications. The modelling of groups within AMIGO render this approach of supporting group communications a source for basic requirements for the GMS.

Bannon and Page [2] have developed a service which is used to support collaborative interaction among groups of users. This service fulfils most of the requirements given in section 2, in particular, it is flexible enough to handle users and groups without restrictions. It is also possible to specify access rights, to use size control for groups, and to distinguish between three types of users for each group, which are administrators, members and guests. However, there is one restriction which limits the usefulness of this approach. It is not possible to store any application-specific data for users or groups. Consequently, the issues of generality and the second view of flexibility as described in section 2 are not satisfied. However, this paper defines the most general and powerful model we have found.

Altenhofen et al. [1] describe a system which is designed to support multimedia conferencing in a heterogeneous environment. The conference directory (CD) is the central database used to register users and groups. It is accessed by using a special CD access protocol (CDAP). The OSI directory service is used by the CD to store and distribute the information concerning conferences. The definition of users and groups are somewhat restricted, eg groups cannot be nested. There is also no possibility to store application-specific data. Consequently, the CD itself does not suit our needs, but the overall architecture of the collaboration environment with its different components and access protocols is interesting.

DePaoli and Tisato [5, 6] describe a general model for specifying and designing conferences. Some aspects of the model of coordinators as basic building blocks are very interesting, especially group mapping and a coordinator's invariants. These two concepts are used to specify interdependencies

between groups such as sub-/superset relations and properties of groups such as limited cardinality. But because of the tight connection between group management, communications, and user interface issues, this model fails to fulfil some of our requirements (such as simplicity and the generality to be used by a large range of applications).

4 Model

So far, we have described the requirements for a GMS and the shortcomings of existing work in the area of designing a group management service. In this section, we will define a model which is appropriate for the problem of a GMS and which is based on object-oriented methodology. The model of how the service should fit into a supporting environment is depicted in figure 1. Other common application service elements (CASE), which might use the GMS, are supporting the application which resides on top of this service layer. The whole set of application service elements (including the GMS) might be viewed as constituting a support platform for groupware applications. According to Navarro, Prinz and Rodden [9], other areas of support (covered by different CASEs) could be services for the support of communication or services for the support of activities. The GMS could be seen as a service for the support of information sharing.

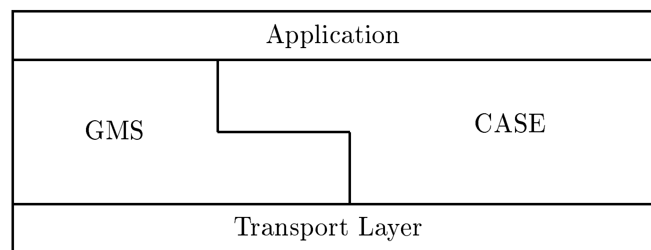

Figure 1: General model of GMS usage

Starting from this general model, the interface between the GMS and the application (which could also be used by other CASEs) has to be defined. Using an object-oriented approach, classes and functions (or methods) have to be defined which could be used by applications using the GMS.

Users and groups are both entities, ie they constitute the set of object types which are used to construct the database. The complete set of users and groups is therefore called the entity space. The functions of the GMS can now be classified by the entities they operate on and by their effect on the entity space. Table 1 shows the functions of the GMS and how they are classified.

	Informational	Modifying entities
Concerning only one entity	get_members get_supergroup get_description authenticate	modify
Concerning the entity space	list search	create/delete join/leave

Table 1: Functions of the GMS

The functions listed in this table are easy to explain. *get_members* is a function which can be used to query all members of a group. Members of a group could be either users or groups, ie any entity. *get_supergroup* is used to query the group of which the current entity is a member of. *get_description* returns the description of the current entity, eg a comment, modification times and other administrative information. *authenticate* is a function used to prove the authenticity of a user. *modify* modifies the

current entity, but only with respect to information which concerns this entity alone. *list* can be used to query the GMS for a list of entities, whereas *search* searches the entity space for a set of entities which satisfy a given condition. *create/delete* are used to create resp. delete entities from the entity space, whereas deletion is only possibly if it does not corrupt the entity structure (see figure 3 for an example). *join/leave* are used to add members to or to remove members from groups.

These functions have to be assigned to classes which make up the model of abstract data types which are available for applications using the GMS. Figure 2 shows the class hierarchy of the GMS with an example of how it can be extended by applications. The classes defined by the GMS are gms, entity, user, and group. The other classes are not part of the core GMS model. Section 6 will give the detailed explanations for these extensions.

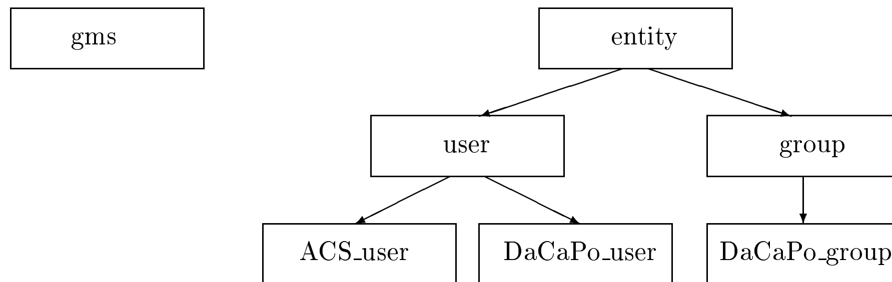

Figure 2: The class hierarchy used by the GMS

The initial class hierarchy of the GMS is simple. However, applications are allowed to define derived classes (ie subclasses) of these base classes to implement application-specific properties of users and groups. For example, ACS_user, DaCaPo_user and DaCaPo_group may be classes which are designed to support specific applications. However, since the derivation (ie subclass) relationship in object-oriented methodology describes the is-a relationship between classes, a DaCaPo_user also is a user. This makes it easy for applications not knowing the DaCaPo_user class to access any object of this class by handling it like a user object.

The following list gives a description of the classes of the GMS. It is not intended to be an implementation description, but should describe the task and the interface of the classes. Since the derived classes ACS_user, DaCaPo_user and DaCaPo_group are not part of the GMS model, they are not described here.

- class gms

  Queries which are related to the entity space are implemented within this class. Starting from the functions listed in table 1, these are the functions *list* and *search*. Every application using the GMS needs exactly one instance of this class.

- class entity

  This class is the base class (ie superclass) of the classes user and group. Although it is never instantiated (ie it is a abstract base class), it serves important purposes. It defines all attributes common to users and groups (such as the name, comments and other administrative information). It is also responsible for some of the security aspects of the GMS, ie authorization is implemented within this class. The class entity furthermore handles the creation and deletion of entities, ie it implements the functions *create* and *delete*. The function *modify* provides applications with the possibility to modify an entity, provided they are authorized to perform such a modification. Every entity object has a list of which groups it is a member of.

- class user : entity

  Users are represented by objects of this class. It is derived (ie a subclass) from class entity. Instances of this class are used whenever users are referenced. Authentication is implemented

within this class, ie it defines the function *authenticate* and stores the information necessary to authenticate a user's identity.

- class group : entity

  This class is used to model groups. It is derived (ie a subclass) from class entity. Instances of this class are used whenever groups are referenced. Functions specific to groups, ie *join* and *leave*, are implemented within this class. Furthermore, this class stores some information which is important for groups, such as the initiator of a group, a list of managers (users which have special rights concerning groups they are managing), a maximum number of members a group can have and a flag which is used to specify whether members may be entities (ie users and groups and their subclasses) or only users and any derived classes. Furthermore, every group object stores a list of all of its members.

To illustrate this class hierarchy, figure 3 shows an example of how an actual (ie object) hierarchy of entities may look like. Here we assume the class hierarchy depicted in figure 2, ie we assume the existence of application-specific classes which are derived from the original GMS classes.

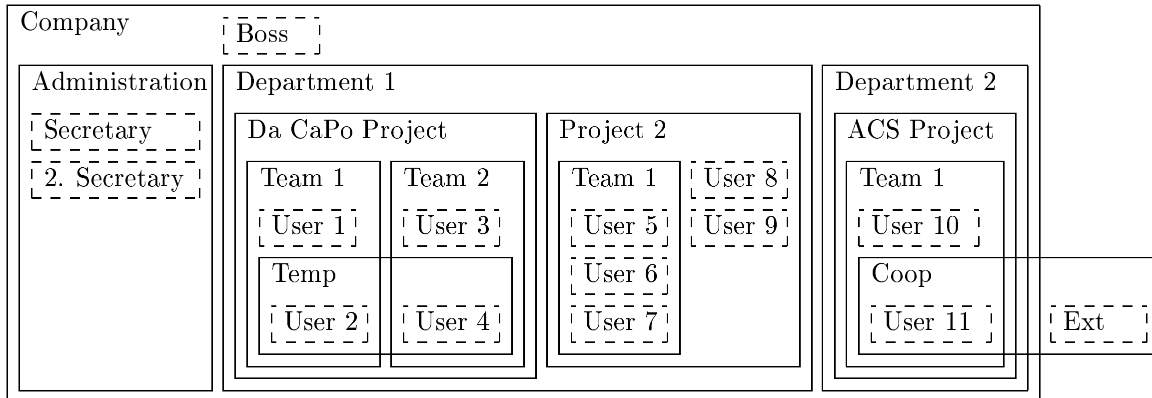

Figure 3: An example of an entity hierarchy within the GMS

In this figure, solid boxes represent group objects while dashed boxes represent user objects. In both cases, it is also possible that a derived class is used instead of group or user itself. If one box is painted inside another, this depicts a has-a relationship between these two objects. For example, the administration has two members (which are both users), namely the secretary and the 2. secretary. Within the Da CaPo project of department 1, there are two teams. both teams are modelled as groups, which may be DaCaPo_groups. However, although this special group may only be meaningful in the context of the Da CaPo project or department 1 (because its implementation is only available there), these teams (and the users they include, which may be DaCaPo_users) can also be used from the administration, which is simply not able to use any special functionality added by the classes DaCaPo_group and DaCaPo_user.

Project 2 of department 1 demonstrates that both users and groups may be members of a group. It would now be possible to *create* a new group within project 2 and to use this group for users 8 and 9 or to just *join* them to team 1. Another possibility is demonstrated by users 2 and 4, who are not only members of their teams 1 resp. 2, but also members of group Test, which may be a temporary group to test new software. Because membership of group Test should not be limited to members of one team, it is made a subgroup of the Da CaPo project. This can be seen as a general rule. A group should be made a subgroup of the group which contains all potential members. This way, membership is limited to potential users instead of allowing everyone to join this group (provided he is authorized to do that). As a last example, group Coop should be discussed. Its members are user 11, who also is a member of team 1 of the ACS project, and user Ext. This user represents an external user (one who does not

belong to the company). Because he is not within the group company, group Coop could not be made a subgroup of group company or one of its subgroups.

5 Implementation issues

After having discussed the model of the GMS, this section will describe some of the implementation issues. Whenever it comes to implementation details, we will use C++ terminology, which is best described by Stroustrup [14]. Figure 4 shows how the GMS will be implemented.

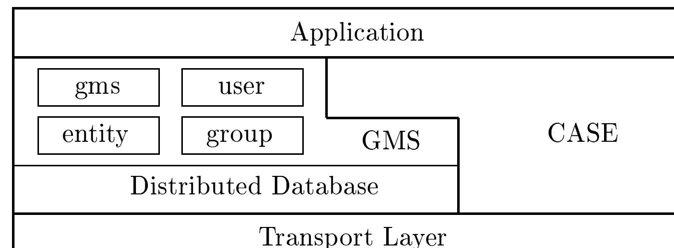

Figure 4: Implementation of the GMS

The four classes of the GMS described in section 4 will be implemented on top of a distributed database. The database will be used to manage the distribution of information and to maintain the consistency of the distributed data of users and groups. Class gms will be implemented as a normal class, whereas class entity will be an abstract base class. Some of the functions which are required for both user and group but which will have different implementations in both classes, eg the function *get_description*, will be declared as pure virtual functions in class entity (this will also make it impossible to create objects of class entity).

The classes user and group will be implemented as derived classes of entity. Within these classes, the functions specific to users or group will be implemented, eg authenticate (user) or join/leave (group) as well as the virtual functions defined in class entity. Objects of the classes user and group will be the objects which most of the time are used to work with the GMS.

It is not quite clear until now which database will be used for the implementation. There are several aspects which are of interest for us. The following list gives an impression of what properties are desired.

- Transport layer

  The GMS will provide a distributed service. To accomplish this, it is required for the different instantiations of the GMS to communicate over a network. Given the architecture depicted in figure 4, the distributed database will be the component which actually accesses the transport interface. Because the GMS should be useable in a heterogeneous environment, it is important for the database to use a standard transport interface or to be adaptable to different transport interfaces.

- Modification notification

  Because of the distributed nature of the GMS, it is possible for two applications (which are at different locations) to have two instances of an entity which represent the same entity. If one application modifies this entity, the distributed database should propagate the changes to all remote sites. However, the information stored within the other entity object may be obsolete because of the modifications. There are two ways to avoid inconsistencies like this one. One solution is to let every entity query the database whether the related database entry was modified or not. This is inefficient and therefore should be avoided. Another solution would be to let the database notify entity objects whenever a modification was performed. This would require entities

to check in and check out with the database, making sure that notifications are only sent when necessary. This could be easily realized using constructors and destructors.

- Scalability

  The issue of scalability as described in the requirements in section 2 is directly related to the scalability of the database being used. It is only possible for us to use a database which is scalable to large domains.

- Schema extensibility

  Because of the possibility to use derived classes instead of the classes user and group, it is necessary for the database to be able to store data which may be of different types (depending on which attributes the derived classes need to store). This could either be accomplished by extending the schema for every new derived class or by defining attributes which are able to store the additional information. This could be implemented by defining a list of pairs which are constructed of a type (eg DaCaPo_user) and its associated attributes.

There are several possible databases which might be used for the GMS. One very popular candidate is the directory as described in the CCITT's X.500 series of recommendations [4]. However, this database is neither very efficient nor is it capable of modification notifications. Another database which might be applicable is NIS+ as described by Winsor [16]. A problem with NIS+ might be the scalability because it is intended for use within enterprises only. Because of the object-oriented approach it is also important to take object-oriented databases into consideration. There are not many distributed object-oriented databases available at the moment. One system in the ITASCA database, which is the successor of the ORION database as described by Kim et al. [7]. It is not clear at the moment whether this system satisfies our requirements.

Since the implementation will be designed for use with different applications, we will be able to test it with "real" applications soon. Two of them are presented in the following section, which describes some of the possible uses of the GMS.

6 Applications

This section will illustrate the concepts described in the last two sections by giving two sample applications of the GMS service. Both examples describe ongoing work in our laboratory. Both projects will use the GMS as part of the services they offer to applications. It should be noted that, despite of the fact that both projects are not explicitly implemented to cooperate, it will be possible to share the GMS information they generate. Figure 2 shows how the classes defined by these two applications could fit into the class hierarchy of the GMS.

Both descriptions are scenarios of how the GMS might be used. Until now, both projects don't use any user or group management. This makes the application interfaces more complicated than they should be.

6.1 Audio Conferencing Service (ACS)

In our laboratory, several projects work on different topics in order to build tools for the distributed use of applications. In the RACE-Project CIO, the idea of sharing X-Applications by multiplexing the datastream of collaboration non-aware applications is pursued whereas in the ETHMICS project, a new architecture for a multimedia workstation was designed. In the MultimETH project, as described by Lubich and Plattner [8], a multimedia conferencing and editing system was designed and realized on top of standardized protocols (an OSI protocol stack). In all projects mentioned, the need for an audio channel during distributed work (eg conferencing) was stated.

For the MultimETH conferencing and joint editing system, an Audio Conferencing Unit (ACU) was designed and is being built. This specialized hardware provides the base conferencing system with flexible switching and mixing facilities for audio datastreams. The ACU performs the mixing of audio signals coming from and going to all conference participants. A special piece of software, the MultimETH audio conference server, controls the ACU and allows the MultimETH system to access the audio conferencing functionality. Based on experiences gained by the MultimETH project, Burkhardt and Lubich [3] identified the functionality needed to support a wide range of conferencing systems with audio.

This work led to the idea to define and implement a generic Audio Conferencing Service (ACS). Generic in this context means that this ACS not only supports one specific application, but is accessible from different kinds of collaboration aware applications. The ACS is therefore designed to support a group of collaborating users with audio. Figure 5 depicts the architecture of one conferencing application in conjunction with the ACS and the GMS.

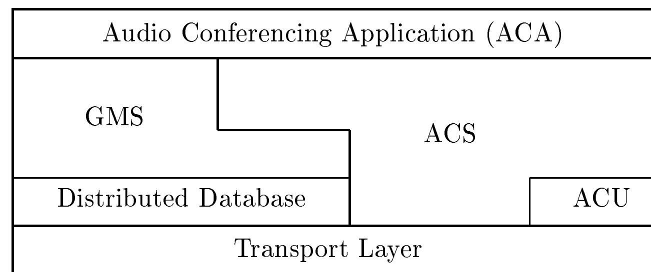

Figure 5: A conferencing application using the ACS and the GMS

The ACS is accessed directly from the application. It sets up and controls the ACU. The ACU does the mixing of the audio signals coming from and going to the conference participants.

The user and group relevant information, eg the phone number of a user, are stored using the GMS, perhaps using a special user class such as ACS_user. Thus, each participant has to be registered with the GMS. The application can use the GMS not only to get the phone numbers of participants, but also to authenticate users or to determine to which groups a particular user belongs. For groups, the administrators name as well as all names of the members contained are held. Conferences must be mapped by the application onto groups. If there is any ACS-specific information which should be associated with groups, a new group class may be created.

One major advantage of using the GMS to keep user and group relevant data is that the data can be accessed from different applications on systems at different locations as well as from the ACS. Thus the GMS frees the application of keeping track of user and group specific data. In addition, the application can operate with the ACS solely by using names and the ACS can get the information needed directly from the GMS.

One possible scenario could be: A user *Chris* registers with the Audio Conferencing Application (ACA). To determine the user's authenticity, the ACA starts a query for user *Chris*. If user *Chris* has been registered already, the GMS returns an object of type ACS_user containing the data concerning user *Chris*. With this object, the ACA can authenticate user *Chris*. *Chris* wants to know which conferences he is a member of and therefore the ACA starts a new query to get this information from the GMS. The GMS returns a list of all groups (a list of group objects) *Chris* is a registered member of. *Chris* chooses to start the conference *test*, and at this time he becomes chairperson of the conference *test*. The ACA stores this information in the group object with the name *test*. Finally *Chris* wants to invite all registered members of this conference. The ACA sends this request to the ACS providing the group *test* as conference specification. This group is used by the ACS as the description of the conference to set up. The ACS itself starts a query for each registered member of the group *test*. For every ACS_user object which is returned from the GMS, it determines the phone number of that member and sets up a phone call on the ACU using this number. Any user or group relevant data which is changed during

the ongoing conference can be saved using the GMS. At the same time, a joint editing system could also access the data of the group *test* and thus start, in addition to the ongoing audio conference, an editing session among the same users.

6.2 Da CaPo

Another possible application of the GMS is the Da CaPo transport system as described by Plagemann et al. [10]. Currently, Da CaPo only supports point-to-point connections, but this will be changed with ongoing work within the project. Current work on multicast suggests that multicast capability will become more important in the future than it is today. Multicast research today is mostly done on protocol levels which are invisible for the user. However, to fully exploit multicasting functions, it is important to have a powerful abstraction on the application level. Figure 6 shows how this can be done in Da CaPo.

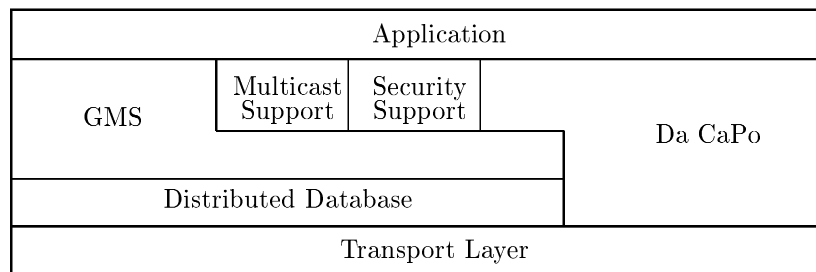

Figure 6: Application of GMS for the Da CaPo transport system

The GMS is used for several enhancements of Da CaPo such as multicast and security support. On the application level, it will be possible for the user to address entities instead of users. Entities will be users or groups. Da CaPo then will use the GMS to find out which connections have to be made. In case of the entity being a group, it will be useful to define a special DaCaPo_group class which has all necessary information, eg multicast addresses. If the application specifies such a group as recipient of a message, the multicast address can be used to transmit the message using the lower layer multicast capability. If there is no multicast address associated with the group (ie it is no DaCaPo_group), the GMS can be used to find out all members of this group. These members can then be addressed individually. This process of using multicast on the lower layers or not, depending on the information available, is invisible for the application which can simply use groups no matter how the members are addressed. This process can even be performed recursively, resulting in the most efficient use of multicast capabilities.

Security support can also be provided by the Da CaPo system in cooperation with the GMS. The GMS can be used to read all information necessary for authentication and authorization of users and groups. If the basic security support of the GMS is not sufficient for Da CaPo, it will be easy to create new classes which contain all necessary features. This should be the main task for applying the GMS in any new environments – creating subclasses of user and/or group which can be used to meet the demands of this specific application.

Another issue the context of Da CaPo is the distributed database used by the GMS. Obviously, multicast is a very interesting concept for distributed databases. It would therefore be interesting to be able to use transport layer multicast for the GMS database. However, the management of multicast groups within Da CaPo is performed by the GMS. The main problem here is which service is used by which other service and how this can be done efficiently. Furthermore, most distributed databases do not use multicast yet.

It can be seen that the GMS is not only of interest to groupware applications, it may also be used for transport services to provide a more abstract service than is usual today.

7 Further work

The next steps in this project are the evaluation for a distributed database and the implementation of the four classes gms, entity, user, and group on top of this database. This implementation will be done using C++ as programming language. The GMS then will be tested alone and in conjunction with the ACS and the Da CaPo project.

A subsequent project will be a model and an architecture for an efficient shared workspace based on the GMS and Da CaPo. Since Da CaPo is specifically designed to support different application needs by dynamically configuring protocols, it is ideally suited to support a service which needs different connections for different data. For example, sharing a video object requires much more bandwidth than sharing a piece of text. The shared workspace to be designed will be able support the sharing dependent of the information type and the user's demands.

8 Conclusions

A model for a group management service (GMS) has been described. The issues of simplicity, flexibility, generality and scalability have been addressed and it has been shown that a GMS model is able to satisfy all these requirements. Simplicity is achieved by using a model of few classes and simple rules of group membership. Flexibility has been addressed by allowing applications to define their own classes of users and groups and also managing this information. Generality is guaranteed by giving applications the freedom to define the semantics of groups, ie groups may be used for many purposes. Scalability can be assured by using a distributed database which is able to be used in a large scale. The application of the GMS within two research projects will demonstrate its usability for different applications.

References

[1] Michael Altenhofen, Jürgen Dittrich, Rainer Hammerschmidt, Thomas Käppner, Carsten Kruschel, Ansgar Kückes, and Thomas Steinig. The BERKOM multimedia collaboration service. In *Proceedings of ACM Multimedia 93*, pages 457–463, Anaheim, California, 1993. ACM Press.

[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] Christoph Burkhardt and Hannes Lubich. Audio support for synchronous cooperative work. To be published in *Proceedings of the 1994 Conference on Upper Layer Protocols, Architectures and Applications*, Barcelona, Spain.

[4] CCITT. The Directory – Overview of Concepts, Models and Services. Recommendation X.500, 1988.

[5] 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.

[6] 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.

[7] Won Kim, Nat Ballou, Jorge F. Garza, and Darrell Woelk. A distributed object-oriented database system supporting shared and private databases. *ACM Transactions on Information Systems*, 9(1):31–51, 1991.

[8] Hannes Lubich and Bernhard Plattner. The MultimETH conferencing and joint editing system. Paper submitted to *Journal of Collaborative Computing*, 1994.

[9] Leandro Navarro, Wolfgang Prinz, and Tom Rodden. Towards open CSCW systems. In Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems [12], pages 4–10.

[10] Thomas Plagemann, Bernhard Plattner, Martin Vogt, and Thomas Walter. A model for dynamic configuration of light-weight protocols. In Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems [12], pages 100–106.

[11] Wolfgang Prinz and Rolf Speth. Group communication and related aspects in office automation. In *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems*, pages 207–223, Munich, 1987. North-Holland.

[12] *Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems*, Taipei, Taiwan, 1992. IEEE Computer Society Press.

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

[14] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, second edition, 1991.

[15] 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 [12], pages 57–63.

[16] Jane Winsor. *Solaris Advanced System Administrator's Guide*. Ziff-Davis Press, Emeryville, California, 1993.

[17] Jane Winsor. *Solaris System Administrator's Guide*. Ziff-Davis Press, Emeryville, California, 1993.