Transport-Independent Group and Session Management for Group Communications Platforms

Erik Wilde, Bernhard Plattner
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH Zürich)
CH – 8092 Zürich

Abstract

With more and more computers gradually changing from isolated, personal tools to networked workstations, group communications is an area of research which has received much attention recently. This paper focuses on a model and the architecture of a system which supports group communications by providing group and session management functionality. The system architecture is related to DNS or X.500, however avoids their complexity by focusing on group and session management and adding functionality where necessary. New functionality is needed for the dynamics of group communications (members of a connection may change over the lifetime of the connection) and increased complexity of relations which may be established between objects. A model is described which defines six object types which represent the relevant objects. Users and groups represent real world users and their relations. Sessions and flows describe ongoing group communications. Flow templates and certificates provide mechanisms for management and security issues. The architecture presented in this paper can be used for group and session management support within different group communications platforms. A description of the implementation as well as implementation results are given in the last section.

1 Introduction

Over the past years, surveys by various authors [13, 29, 31, 37, 51] have shown the increasing need for multipoint communications and the requirements for platforms supporting this type of communications. Additionally, the requirements of transporting multimedia data has also been applied to multipoint communications [6, 12, 20, 44], leading to multipoint multimedia communications. One common approach for designing a platform which supports multipoint multimedia data transport services is the separation into different planes, one dealing with the actual data transfer and another being responsible for management issues (examples for such a design have been described by Campbell et al. [10] and Banerjea et al. [3]). This paper presents an approach to the management plane which is based on the assumption that many problems to be solved for multimedia group communications are independent from the transport infrastructure being used and should therefore be addressed inside a component which is reusable for different transport platforms. This approach has the following advantages:

- *Reduced implementation costs.* Because of the transport-independency of such a component, it could be used in different transport platforms without having to implement the functionality for each platform. This leads to a reduction of implementation costs for new platforms using this component.
- *Transport-independent and application-independent naming.* The name to address mapping is one of the tasks of the management plane. If naming is implemented by a transport-independent component, it is possible to use the same name space in all applications, independent of the address space imposed by the underlying transport infrastructure. This would also eliminate the situation of today, where each collaborative application has its own name space so that the use of

more than one collaborative application can become a very complicated process in terms of user and group management.

- *Session directory functionality.* A session directory similar to the well-known mbone session directory would be possible, and it would be a more general directory. It would not only list the sessions and the respective applications, but also the transport infrastructure being used. This way it would be possible to implement collaborative applications which are able to use different transport infrastructures while using the same abstractions for group and session management. Furthermore, it would be possible to support user identification, authentication, and authorization in a consistent way for all applications.

This paper illustrates that there is functionality which can be clearly separated from transport-dependent issues and hence be implemented in a separate and reusable system. Basically, what is needed for group and session management is a distributed database. However, since we restrict the usage of this database to the support of group communications platforms, we can use a design which takes into account the characteristics of this specific application area. The management of group communications must encompass users, groups of users, the communication between users, and provide a general model which makes it possible to establish relations between these entities. An example is a collaborative editor, where the application itself (ie the editor) only wants to deal with abstract entities representing users, groups of users which are authorized to read and/or modify a document, and communications between users. In this context, participation in a communication should only be possible if the user is sufficiently authorized. The information needed for this decision could be stored in a group and session management system, which could then be used by the editor without the need to implement authentication and authorization. This information could then easily be used by other applications, eg a drawing tool, which could use the same groups for authorization and therefore would share this data with the editor. As a consequence, all data necessary for managing collaboration (ie for identification, authentication, and authorization) would be managed by the group and session management system and could therefore be accessed by all applications using this system. The access would be possible through a special component communicating with the group and session management system.

A group and session management component is used for different tasks within a group communications platform. These tasks may be divided into security issues, name to address mapping, Quality of Service (QoS) issues, and the core functionality, group and session management. The following paragraphs discuss these issues in greater detail.

- *Security issues* are addressed in the form of identification, authentication, authorization, and certification. These are the security issues which are important in the context of group and session management and the provision of certified software components.

- *Name to address mapping* is the functionality which enables users of any communications system to use names instead of addresses, ie to use an abstract form of naming entities (communication endpoints). This makes it possible to use logical names independent from locations and connections which are mapped onto addresses which can be used for addressing in a given transport infrastructure. Saltzer [39] describes the concept of naming, addressing, and binding for the Internet.

- *QoS issues* are important when speaking of actual connections. The well-known multimedia QoS parameters like throughput, error rate, delay, or jitter, can be seen as one special case of a general characterization which may be defined for each connection. The parameters being used should be as versatile as possible to allow the utilization of yet to be defined types. Szyperski and Ventre [45] and Carle et al. [11] describe this concept of QoS parameters for multipoint communications in greater detail. Security issues may also be addressed for communications itself via QoS parameters, where authentic and/or private communications may be required.

- *Group and session management* is necessary for making the naming space (ie the names which may be used) flexible and configurable. Typically, groups of users are changing over time, so it must be possible to create new groups, to change group members, and to delete groups. Furthermore, the communication inside a group (which we call a session) should also be manageable by users of the communications platform. Another important functionality in this area is the provision of authorization for group and session memberships and modifications.

The rest of the paper will describe this functionality in greater detail and describe its modeling and implementation. Section 2 gives a short overview over other work in this area and related activities. Section 3 describes our model and the architecture of a system for group and session management. Section 4 gives a short overview over implementation issues. Section 5 concludes the paper and describes some future work.

2 Related work

Group communications has become a very active research area in the last years. However, most of the work concentrates on multicast protocols (ie the task of transmitting data over a network to more than one receiver without wasting network resources), especially multimedia multicast protocols, and high-level issues, such as toolkits for programming distributed multimedia applications. Developments in the area of group and session management for group communications frameworks are not so widespread[1], although there are research activities and also standardization efforts. The rest of this section will describe some of these projects. A more detailed description has recently been published by Mauthe et al. [31].

2.1 Research activities

The BERKOM Multimedia Collaboration Service (MMC) described by Altenhofen et al. [2] is an example for an application oriented development of a communications platform. MMC is a part of the BERKOM multimedia teleservices, which also include a Multimedia Mail Service (MMM) and a Multimedia Transport Service (MMT). MMT is a pure transport service and does not offer any group management services. It is based on a variant of XTP which is called XTP-Lite. MMC contains various components, of which Conference Interface Agents (CIAs), Conference Managers (CMs), and a Conference Directory (CD) are relevant for the scope of this paper. Communication between these components is realized by two protocols, the CD access protocol (CDAP) and the CM access protocol (CMAP). These protocols are RPC-based and built on top of the ISO ROSE mechanisms. The architecture of MMC is similar to the architecture described in this paper. However, the service provided is application specific (restricted to conferencing systems) and therefore can be regarded as one possible application of the general concept of a group and session management service described in this paper.

GCommS by Mauthe et al. [32, 36] is a project at Lancaster University which is concerned with the support of group communications, especially for multimedia data. One of the components of the architecture proposed for GCommS is the QoS architecture QoS-A described by Campbell et al. [10]. The transport aspects of GCommS are defined in detail, but the group management services are only described on a very abstract level. Currently, no specification is available, although a transport service (the M-Connection service) has been implemented. GCommS therefore can be regarded as one of the many projects where group management is necessary to provide a well-functioning platform, but is not specified in detail. Other examples for platforms where group management is identified as an

[1] Although we concentrate on group and session management, there are also some aspects of directory services in our system. The relation between our system and the two most popular directory services (X.500 and DNS) are described in section 3. However, the main contribution of our work is a model of abstractions for group communications rather than the design of a distributed directory service. The operation of X.500 has been described by Smetaniuk [43], and a description of the DNS architecture has been published by Albitz and Liu [1].

important component but not investigated in detail are the *x*AMp architecture described by Rodrigues and Veríssimo [38], PRISM by the Toutains [46], or Tenet described by Ferrari et al. [7, 15].

CIO multi-peer communications as described by Henckel [21, 22] is a very interesting concept in terms of functionality. The transport group management defined for the CIO transport service has many similarities to the model we describe in this paper. A user of the CIO multi-peer transport service uses two different components for accessing the transport service and the transport group management service. Communication is handled with two completely separate protocols. However, CIO transport group management is limited to one communications platform (ie depends on the usage of the CIO transport service), and it has not been implemented. Furthermore, since the X.500 directory service is proposed as a basis for the transport group management service, it will be impossible to have notifications sent to users. Furthermore, X.500 tends to be slow since it has been designed as a service which is often used for reading and rarely for changing information.

2.2 Standardization bodies

In the Internet world, work is going on in the Multiparty Multimedia Session Control (MMUSIC) working group of the Internet Engineering Task Force (IETF). One result of this work is a description of the Internet multimedia conferencing architecture by Handley et al. [16], which is shown in figure 1. The component of this architecture which is most relevant for our work is the session directory, which is based on the Session Description Protocol (SDP) v2 described by Handley and Jacobson [17]. However, since SDP can only be used for session advertisement, because it is only used for the distribution of announcements, an additional protocol is required which is used for specifically inviting users to sessions. This protocol is the Session Initiation Protocol (SIP) and is described by Handley et al. [19][2]. This splitting of session relevant information into two separate protocols has been caused by the uncoordinated development of the mbone (a good overview of the mbone is given by Eriksson [14]), which originally was only used for multicasting sessions with a simple session announcement protocol (SDP v1). Since a more powerful support of group communications (at least for tightly coupled sessions) also needs a way to identify users and groups of users, we believe that a different design than the one currently being in use for the mbone is necessary. This would remove the situation of today, where each mbone application has to implement its own management of users, groups, and sessions. From the user's point of view, using an architecture as proposed in this paper, it would not be necessary any more to start each application of a mbone session to see all session participants, but this information could be provided by the session directory (which would have access to the group and session management system).

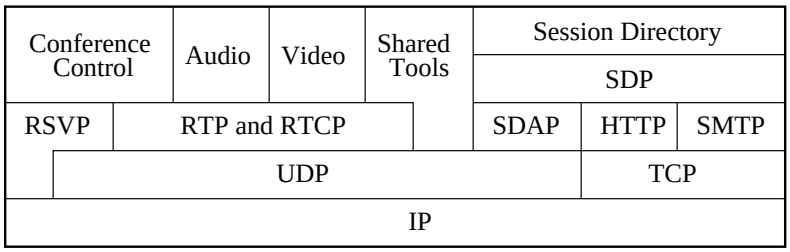

Figure 1: Internet multimedia conferencing architecture

ITU's T.120 series of recommendations [27] is an example for a standardized application architecture which also incorporates group and session management functionality. The basis of the T.120 infrastructure are the network specific transport protocols defined in T.123, which at the moment support data

[2]The Session Initiation Protocol (SIP) is the result of a merger of the Session Invitation Protocol by Handley and Schooler [18] and the Simple Conference Invitation Protocol by Schulzrinne [42], which have been earlier proposals for a protocol for the invitation of users to conferences.

transfer using integrated services digital networks (ISDN), circuit switched digital networks (CSDN), public switched digital networks (PSDN), and public switched telephone networks (PSTN). Extensions to include future broadband networks are under study. T.123 is used by the multipoint communications service T.122/T.125, which defines a network independent service with flexible data transfer modes (broadcast and request/response), multipoint addressing (one to all, one to sub-group, and one to one), and multipoint routing (shortest paths to each receiver and uniform sequencing). Recommendation T.124 defines a generic conference control which uses T.122's multipoint communications service. The abstract services of the conference control include create, query, join, invite, add, lock, unlock, disconnect, terminate, eject user, and transfer services. These services provide a powerful environment for implementing conferencing applications, which use T.124 and T.122 services. However, the applicability of these recommendations is limited because of the concentration on real-time conferencing. The T.120 series of recommendations can therefore be regarded as a specific example which should be kept in mind when designing more general group and session management services.

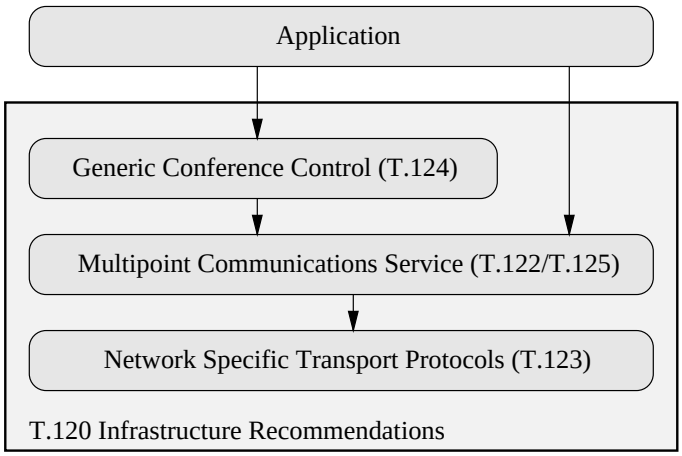

Figure 2: Architecture of the ITU T.120 infrastructure recommendations

In ISO, group communications is dealt with in different committees and working groups. We will only consider the work going on in JTC1/SC6, which is currently the most active with respect to group communications. ISO's multipeer taxonomy [25] (which is also described in a paper by Mathy et al. [30]) gives a general architectural model for group communications, which may be used within different OSI layers. Concepts defined in this taxonomy describe a group (a set of entities), group memberships (entities belonging to a group), population characteristics (a set of attributes a group may have), group associations (data transfer relationships), and conversations (the basic components of group associations). These concepts can be combined to form a model which is very flexible in terms of group communications. However, as with many other standardization activities, no implementations of these concepts exist, so it is not possible to experimentally evaluate the usability of this model. The ISO draft document on an enhanced communications transport service [26] incorporates most concepts described in the multipeer taxonomy. The naming and addressing model defined in this draft is more complex than in point-to-point transport services, because conversations and group associations have to be taken into account.

3 GMS – Group and session management system

Group and session management can be identified as one task which is necessary in every system supporting group communications[3]. Figure 3 shows how a component providing this task could be integrated

[3]The amount of support varies very much between different approaches, but it is generally accepted that there is a need for group and session management. Publications on this topic have been written by Pejhan et al. [34] and Braudes and

into a communications platform (the component is labeled GUA, which stands for GMS user agent). The model is independent from the communications platform being used, although it is mainly influenced by discussions within the Da CaPo [35] and MCF [5] projects. These projects, where the main goals are efficient communication protocols achieved by dynamic configuration, developed the need for group and session management after broadening their model of communications from point-to-point to point-to-multipoint (or, in case of MCF, multipoint-to-multipoint) connections.

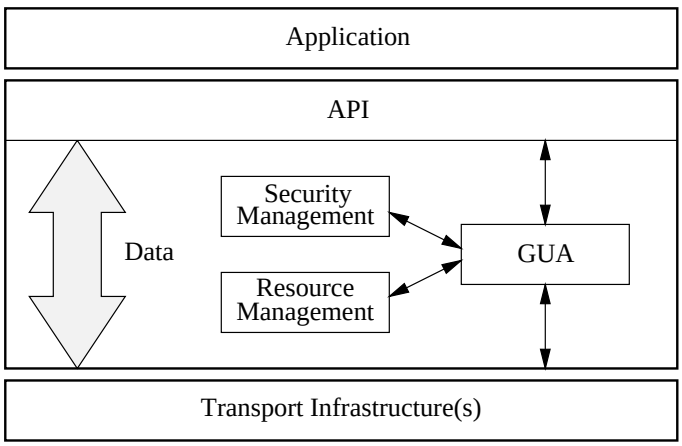

Figure 3: Group and session management inside a communications platform

The architecture we propose for the design of a group and session management service is that of a directory service, where each user of the directory service uses a component which is implementing the access protocol to the directory service. With respect to this design, our proposed group and session management service is similar to the Internet domain name system (DNS) defined by Mockapetris [33], or the X.500 directory service standardized by ISO and ITU [28]. However, because in our case we need more than a pure lookup service (where the directory system, from a user point of view, only reacts and never acts), and special functionality (such as operations for joining and leaving groups and sessions), a new service is required. This new service is provided by the group and session management system (GMS) and is available through a GMS user agent (GUA).

The main difference between DNS and GMS is that DNS lookups are very short connections, while GMS connections exist over the lifetime of a session, and that the typical GMS user has an identity which can be used for additional functionality, while a DNS user has anonymous access to DNS data. The main difference between X.500 and GMS is the availability of object types specifically designed to support group communications, that GMS system agents (GSAs) are able to become active (instead of the purely reactive Directory System Agents), and that the internal GMS system protocol (GSP, described in the following section) is designed for faster propagation of information than DSP (the internal protocol of X.500) by using a multicast transport service.

The general GMS model introduced in the previous paragraphs can be further refined to define an architecture which can be implemented and tested. The following sections describe different topics of this architecture. After a general description of the architecture's main building blocks, the GMS object types are specified. QoS issues are the topic of the last section, where the different steps and definitions of QoS usage are explained.

3.1 Architecture

The architecture of GMS is similar to that of other directory services, in the sense that the GMS service is available through a GMS access protocol (GAP) which is used by GMS user agents (GUA) to access

Zabele [9].

GMS system agents (GSA). The GMS service is implemented by the GMS system protocol (GSP), which is used by GSAs to communicate with each other. The overall view of this architecture is shown in figure 4. In this figure, it can be seen that the connection to GMS (through the GUAs using GAP) is completely independent from the data transmission between the three example participants of a session. The design as a specialized directory service was motivated by the observation that the group and session management functionality we want to provide is very similar (although more complex) to the name to address mapping of existing directory services. Since we need to have a permanent database about users, their properties (such as authentication information), and groups of users, a persistent storage is required.

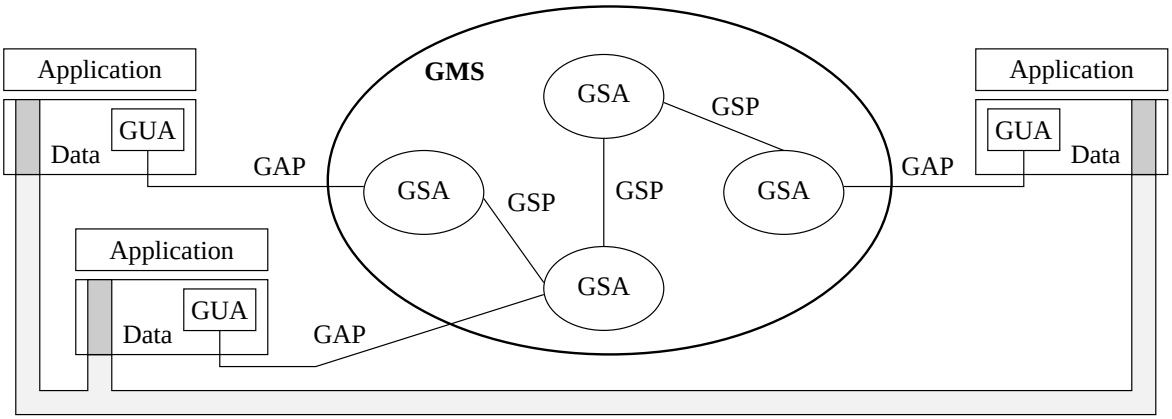

Figure 4: GMS architecture

While this model of GMS is very simple, it has been designed for large scale application and therefore needs to apply techniques which make it feasible to run the system even if it is distributed world wide with a large number of GSAs. One of these techniques is a structured name space which makes it possible to differentiate between local and non-local operations and to find non-local objects in a relatively easy way. This is called the usage of domains. For example, currently DNS comprises[4] about 488000 domains (including 183 top-level domains) with an average of one subdomain per domain[5] and an average depth (ie level of domain nesting) of 2.3. We see these numbers as upper bounds for GMS, since it is not designed to be used by all networked computers, but only by those needing group and session management support. Thus it can be seen that GMS must be able to work even in the large scale, which mainly influences GSP (GAP is almost independent from this scaling issues, only the naming is influenced by the scaling requirements). The two GMS components GUA and GSA can be described as follows.

- *GMS user agent (GUA).* GUAs are the components outside GMS which are used to access GMS. Typically, a GUA is not much more than a protocol machine which implements the GMS access protocol (GAP). As such, the GUA provides an interface which can be used by other components of the communications platform. It performs encoding and decoding of protocol messages and also implements some local parameter and state checking. The GUA is not intended to be directly accessible to application programmers, because the communications platform designers should be free to define the platform's API in a coherent way without having to accept a predefined GUA API for applications.
- *GMS system agent (GSA).* GSAs are the components which, in their entirety, make up GMS. Each GSA is a component of this distributed system and communicates with either other GSAs

[4]These numbers are taken from the latest Internet domain survey, July 1996, produced by Network Wizards (http://www.nw.com).

[5]The number of subdomains per domain varies very much. Toplevel domains have an average of 2056 subdomains, while first-level subdomains have an average of 0.27 subdomains.

(using the GMS system protocol GSP), or GUAs (using the GMS access protocol GAP). In GMS, data storage is distributed, so each GSA may store some local data which can be accessed by another GSA. GSAs are grouped into domains, which form logical sets for data storage and access. Domain are organized hierarchically, with each domain having one superdomain and zero or more subdomains. However, there is no root domain, but a number of toplevel domains, which do not have superdomains, but know all other toplevel domains.

The two main components of the GMS architecture, GMS user and system agents, are communicating using special protocols. These protocols are used for accessing GMS (via GUAs) respectively for communications inside the GMS. The following paragraphs give a more detailed description of these two protocols, which form the other important building blocks of GMS.

- *GMS access protocol (GAP).* GUAs are used for accessing GMS using entry points provided by GSAs. These entry points must be accessed using GAP, which is an asymmetrical protocol. The complete specification [47] contains definitions of used object types (which are described in section 3.2 of this paper), relations between objects, PDU definitions, and state transition diagrams describing the behavior of the two communicating agents.

  GAP itself can be separated into three phases. The first phase (*GUA binding phase*) consists of connection setup and tear-down between GUA and GSA. The second phase is the *user authentication phase*, where a user is identified and authenticated. Authentication is handled in a very generic way, so that weak and strong mechanisms (from anonymous bindings or no authentication to standard Unix passwords or challenge/response schemes with multiple iterations[6]) can be used. After successful authentication, the third phase (*user bound phase*) is entered and services for accessing GMS data can be used. These services include create, modify, query, delete, join, and leave for the appropriate objects. Most GAP requests are initiated by the GUA (ie the user), but there are also some situations where the GSA becomes active (eg when a user receives notifications, invitations, or renegotiations).

- *GMS system protocol (GSP).* GSAs communicate using GSP, which is a protocol using a multicast and a unicast transport service. The complete specification [48] describes for each PDU whether it is sent via multicast or via unicast. Multicast communications has been chosen to take advantage of the domain based naming of GMS objects. Each domain also has a multicast address, which can be used to send requests to all GSAs of a domain. This way, we can achieve faster access to GMS objects.

  GSP operation can be grouped into several areas. Three areas needed for internal purposes are the periodic exchange of domain information between GSAs of a domain, the management of GSA entering or leaving a domain, and the assignment of tokens inside a domain. GSP uses three different tokens to assign roles to exactly one GSAs inside a domain. These roles are the propagation of domain name resolution requests, the creation of objects, and the forwarding and processing of queries[7]. When a GAP request is processed by a GSA, it normally needs to find out the address of the domain where the object specified in the GAP request is stored. This is done using domain name resolution, which is the fourth area of GSP operation. The last area is the processing of GAP operations once the domain address is known. Most GSP operations are performed using a mixture of multicast and unicast transmission. Typically, the request is sent via multicast and the response is given using a unicast connection.

These descriptions of GMS components (GUA and GSA) and the protocols used for communications between these components (GAP and GSP) give an overview of the GMS system architecture. More

[6] The actual authentication mechanisms available for binding are determined by the GSA the user connects to. If a GSA only supports weak authentication schemes, it is not possible to get a strong authentication through this GSA.

[7] A more detailed explanation of the three token types and their usage is given in [50].

complete descriptions of the GMS components and protocols can be found in [49, 50]. The following three sections deal with aspects of the data being stored and exchanged, which are the object types available (section 3.2), the security model which is defined in GMS (section 3.3), and the way QoS issues are handled (section 3.4).

3.2 GMS object types

All definitions of the object types available inside the GMS are specified in ASN.1 and verbal explanations of the semantics. Detailed descriptions are given in the GAP/GSP specifications [47, 48], here we will only give an overview of the object types, their attributes, and the general concept behind each type.

- *User.* A user is a person or entity using GMS. Each user has an identity (a name) and one or more ways of authenticating himself. This authentication may vary from no authentication at all (ie it is sufficient to use the right name) to sophisticated authentication schemes with multiple challenge/response iterations. A user object contains information about a user, such as his real world name, a description, his email address, and a list of the bindings of a user, ie the list of active GMS connections a user has.

- *Group.* GMS groups may consist of users and/or groups, depending on the definition of the group. How to join and leave a group depends on the group's join policy and authentication requirements. Joins and leaves may be notified to a group's managers and/or members. Each group object may contain a group's real world name (eg the name of a company or a company's department), a description of the group, a group's mail address, and the access rights, which determine who is authorized to modify the attributes of the group.

- *Flow Template.* For several applications and communications platforms it is useful to have a number of predefined possibilities to set up connections. Flow templates contain information about data types which may be carried by a flow of that type, the necessary transport service, data which is needed to set up a flow of that type, information about uni- or bidirectional services, and a set of QoS parameters, which can be used to give a description of the flow template. However, flows may also be created without using a flow template.

- *Flow.* A flow is one data flow for data transport. Depending on the flow's definition, it is either uni- or bidirectional[8], has a limited number of senders and/or receivers, and a renegotiation policy, which determines who is authorized to initiate QoS renegotiations for that flow. Flows are created when a session is created and are deleted when a session is deleted. It is not possible to add flows to or remove flows from a session after it has been created. Joining a flow takes place when a session is joined, and a flow is left when the session of a flow is left. Flows can be based on flow templates, but it is also possible to create flows without using a flow template.

- *Session.* The main metaphor for group communications is a session. Each session is used to logically group a number of flows and to create an abstraction for management, authorization, and admission control for flows. The flows of a session are created when the session is created and deleted when the session is deleted. When joining a session, not all flows of the sessions must be joined, so users can choose which flows to use[9]. Sessions may have application specific information, which consists of an application identification and application specific data, which may be interpreted by the application. Furthermore, the duration of a session may be given with

[8] Unidirectional flows can only be joined as sender or receiver, whereas bidirectional flows can also be joined as sender and receiver.

[9] An example for this is a conference session including an audio and a video flow. Some users may, due to bandwidth or local hardware limitations, choose to only join the audio flow.

either start or end times or both. In addition, it is possible to specify which authentication level a user must have to successfully join a session (provided he is authorized sufficiently). Authorization is based on the session's join policy which may be open (everyone may join), group (only members of the group associated with the session may join), or managed, which may be either relative (a given percentage of managers must confirm) or absolute (a given number of managers must confirm).

- *Certificate.* Applications with special security requirements may have the need to store certificates inside the GMS, which are used for validating data identity and integrity. Certificates include the type (which may be a predefined type or any other type), the name type (which also has a number of predefined values and the possibility to define own types), the certificate's validity, a simple name, and data and signatures, which contain the information which is necessary for validating the data.

These object types are available for usage inside the GMS and may be created, modified, and deleted using GAP services. However, because a number of attributes and relations (such as the addressing information attribute and the senders relation of flows) needs to be modified according to their semantics, they are modified only implicitly by using GAP services (such as the *join session* service, which adds a sender's address to a flow he joined and modifies the senders relation accordingly).

3.3 Security model

Security in GMS is used in three different ways. The first way is the provision of authentication mechanisms for users, which make it possible that user identities are verified using different authentication methods[10]. The second way of security provision is the authorization which is supported by all object types via an access policy concept[11]. The third way of security usage in GMS is the support for the secure distribution of software components. This is achieved by using the certificate object type described in section 3.2. The first two ways of GMS security are described in the remainder of this section.

The security requirements for group communications vary from no security at all to highly confidential meetings. In order to support this variety of security requirements, GMS defines a security hierarchy which is used to provide each application with the security level it requires. There are three main issues regarding the security hierarchy.

- *Authentication levels.* Each user has to identify himself when using GMS. This identification is based on the user's name. Additionally, the user must give a proof for his identity, ie he must use an authentication mechanism. Depending on the strength of the authentication mechanism, it is more or less easy to cheat and to claim a false identity[12]. GMS therefore introduces authentication levels which are used to identify the strength of the authentication mechanism a user used when claiming his identity.

  A user can have multiple authentication methods which are stored in the user object. For example, a user can have methods such as Unix passwords when normally using the system and stronger authentication mechanisms whenever he takes part in confidential meetings. The user therefore can choose the authentication level each time he works with GMS.

- *Authentication requirements.* This information is the counterpart of the authentication level defined by the method a user used to authenticate himself. Each GMS object defines authentication

[10] This can be compared with the well-known login procedure of operating systems, where the user has to type in a password to give a proof of his identity.

[11] This is comparable to the concept of file permissions used in operating systems, which define the operations which are allowed for certain user classes for every file.

[12] For example, when the algorithm for the encryption of the passwords in Unix password files was invented, it was almost impossible to break it. With today's high-performance workstations, it only takes a few hours to break this encryption.

requirements. These requirements define the necessary authentication level which is needed to access the object. Only if a user has used an authentication mechanism as least as strong as defined by the authentication requirements of an object, he is given access to the object.

- *Access policies.* Naturally, authentication is not sufficient for gaining access to an object. If a user satisfies the authentication requirements of an object, the object's access policy is used to check the users authorization. Access rights for objects are defined for reading, modifying and deleting an object. It is important to remember that the access policy of an object is only used if the object's authentication requirements are met.

Consequently, access to GMS objects is based on the three elements authentication level, authentication requirements, and access policy. Only if all these requirements are satisfied according to the individual settings, the access to an object is granted. GMS supports predefined authentication levels as well as a method to define new authentication levels which can be defined according to the needs of applications which do not want to use the predefined levels.

3.4 QoS issues

QoS is a very important aspect of multipoint communications, especially if multimedia is also taken into account. GMS must be able to support QoS aspects in a general way which makes it suitable for the support of multimedia multipoint communications. Two main issues can be identified here, the QoS parameter types (defining how QoS can be defined) and the procedures available to manipulate parameters of these types (defining how QoS can be applied).

GMS has four QoS parameter types, which are *unsorted values*, *sorted values*, *integer values*, and *real values*. Unsorted Values are a set of predefined values, which are not in any particular order (an example for this is a QoS parameter which defines a coding algorithm, where it is not possible to arrange the different algorithms in any order). Sorted values are also predefined values, but these values are given as a sequence, because it is possible to arrange them in an order (an example for this type of QoS parameter is the selection of an error detection algorithm, which may be given as a sequence of none, CRC8, CRC16, and some more sophisticated algorithms). Integer and real values represent the two basic types of numbers which may be used (for example for throughput or error probability values).

Any parameter defined for a flow template or flow may be of one of these types. The interpretation of the parameter's values depends entirely on the parameter's name. It is therefore important for GMS users to agree on names for QoS parameters to avoid misinterpretations. For this reason, a number of predefined QoS parameters is available, where the semantics of each parameter are clearly defined. Additional parameters may be used at any time, although it is strongly recommended to use the predefined types whenever possible.

The second aspect of QoS, as mentioned above, are the procedures available for manipulating QoS parameters. Given the GMS object types (as described in section 3.2), four different events where QoS come into play can be described.

- *Flow template creation.* When a flow template is created, it may be given a number of QoS parameters. Depending on the transport infrastructure, the number of parameters already necessary or even known at this point in time may differ a lot. For the different types of QoS parameters, values and/or limits may already be defined, if possible.

- *Session creation (flow creation).* Every data transfer is represented by a flow, which is created when the session it is a part of is created. Each flow's QoS parameters are defined by their names and types and at least a default value (which is the value used for joining participants if no local modifications are requested). Optionally, a weakest limit for joining the flow and strongest and weakest limits for renegotiations may be defined. For unordered values QoS parameters, there is

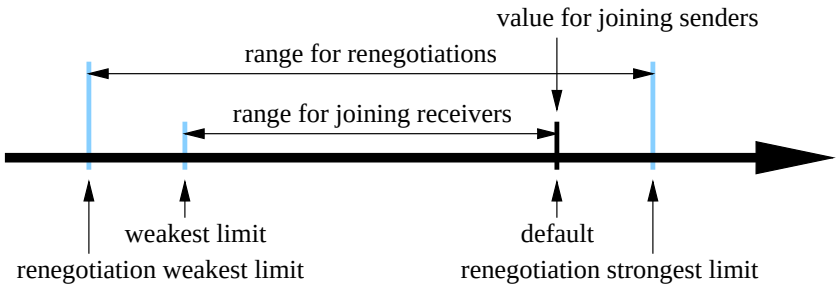

Figure 5: Values and ranges for QoS parameters

no order of the values, therefore the join and renegotiation values must be given as sets of values. The concept of QoS parameter values and ranges is shown in figure 5.

- *Session joining.* When joining a session, the QoS parameters being used are taken from the flow's QoS definitions. According to the user's requirements, a weaker value than the default may be selected, as long as it does not fall short of the weakest limit for joining the flow. However, this is only possible for receivers, senders must always join with the QoS value given as the default value. This is done to make sure that data being sent on a flow is always of a well-known quality, because only receivers are allowed to perform down-grading. Because the actual QoS establishment lies outside the scope of GMS (which is only responsible for storing the values), it is required that transport infrastructures using GMS control the QoS parameters according to the values provided by GMS.

- *QoS renegotiation.* QoS renegotiation is the process of defining new default values and weakest limits for QoS parameters of any renegotiable flow. This renegotiation may be limited by the renegotiation limits of a QoS parameter, if present. All participants (senders and receivers) of a flow are informed of QoS renegotiations of that flow.

A detailed and complete description of the QoS parameters and their manipulation along with the ASN.1 definitions can be found in the GAP specification [47]. However, it should always be kept in mind that QoS establishment and renegotiation are tasks to be performed outside GMS (by the transport infrastructure or the application), while GMS is only used for distribution of QoS values and renegotiation notifications. The GMS QoS model is fairly general, although it is possible that the QoS model of a transport infrastructure would not fit into the framework given by GMS. However, we believe that the model of GMS is general enough to cover the QoS concepts of present and future transport infrastructures.

4 Implementation and results

The implementation of GMS (ie of a GUA and a GSA) is based on Sun workstations running Solaris 2.5.1. For generating code to encode/decode ASN.1 data structures, we used the Snacc ASN.1 to C/C++ compiler from the University of British Columbia, Vancouver [40]. The code produced by this compiler is rather large, but it is fast [41]. As unicast transport protocol for GAP and GSP we use TCP/IP. As multicast transport protocol for GSP we use the multicast protocol described by Bauer et al. [4], which provides a reliable, FIFO ordered multipoint-to-multipoint transport infrastructure.

4.1 GUA implementation

The GUA implementation focuses on two aspects, the first being the easy adaptability of the GUA code to another transport infrastructure. We therefore used only abstract procedures for accessing the

transport infrastructure and created a network adaptation layer which can easily be modified as long as the new transport infrastructure being used provides reliable, connection-oriented communications. This design can be seen in figure 6.

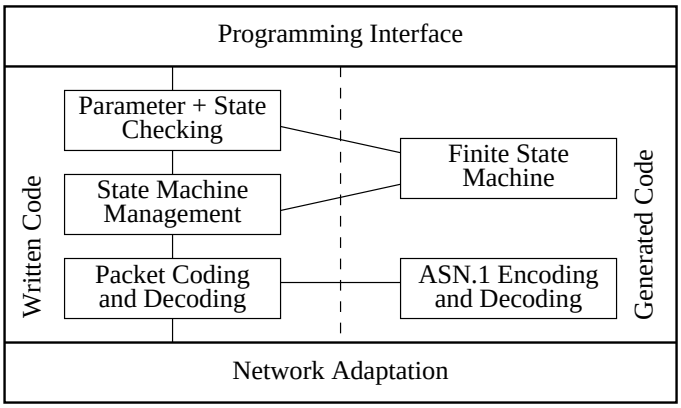

Figure 6: GMS user agent (GUA) design

The other important aspect is the one of control flows. The GUA can get input from both the user (at the programming interface layer) and the network (at the network adaptation layer). Because we did not want to delay processing of one direction until the other direction has been handled, we use a multi-threaded design, where one thread is responsible for processing requests from the programming interface layer, while the other thread processes requests coming from the network adaptation layer[13]. The reason why we decided to use this solution is because it is possible that multiple users use one GUA (which is part of the communications platform), and it is important to make sure that no user is able to interfere with other users' work with GMS.

The implementation of the GUA component consists of seven main components. Network adaptation and programming interface are cleanly separated to make the code easily adaptable to different environments. Furthermore, we use generated code for the GUAs finite state machines and the encoding and decoding of GAP PDUs. This code is controlled by three components, which are responsible for checking parameters and the state of the GUA's state machine, the state machine management (ie performing state transitions and executing the appropriate actions), and the marshaling and unmarshaling of arguments.

Given these parts of GUA code, we implemented the GUA as a library which can simply be linked to any program which wants to use GUA functionality. The only restriction is that the resulting code will be multi-threaded because of the GUA design which includes multiple threads. Apart from that, there are no special things to be considered when using the GUA library.

Although the resulting code is quite big (currently, the library has the size of half a megabyte), which mainly is the result of the code generated by the ASN.1 to C compiler, we observed that the execution times of GUA procedures are small. This is the result of snacc generated code, which is optimized with regard to time and not space. However, the size of the code can be a problem when the system it is run on does not have enough real memory. The paging activity then is a much bigger performance problem than the CPU time required for executing the code.

4.2 GSA implementation

The GSA implementation consists of three main blocks, which are the implementation of the two protocols GAP and GSP, and the internal logic, which is responsible for performing the actions which are requested by either a GUA or a GSA. It is also possible to have GSAs which do not provide

[13]For each incoming request, a new thread is created which then calls the user-provided call-back routine to handle the request. Upon termination of the call-back routine, the thread exits.

GAP access. In this case, the block containing the GAP functionality is omitted. This architecture is depicted in figure 7, which goes into a little more detail. In this figure, each labeled block represents a process, while the MQ components are Unix message queues, providing a convenient interprocess communications mechanism[14]. The dotted boxes contain the three blocks mentioned above.

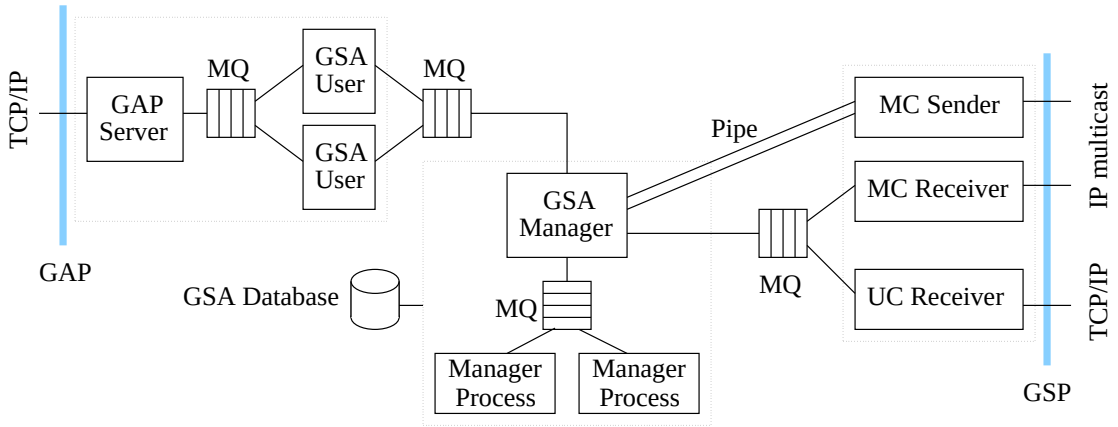

Figure 7: GMS system agent (GSA) design

The main component of the GSA is the GSA manager which is the process started first during initialization. If the GSA supports GAP, the GSA manager starts the GAP server, which is then able to accept connect requests from GUAs. The GAP server currently uses TCP/IP, but it could be easily extended to support other transport protocols as well. The GSA manager also starts multicast processes for sending to and receiving from multicast groups. These processes implement the reliable multicast protocol mentioned above. Furthermore, the GSA manager starts a unicast receiver process which handles all incoming unicast requests for the GSA. These are replies to multicast requests. The multicast modules are implemented on top of IP multicast, while the unicast receiver is based on TCP/IP[15]. Finally, the GSA manager is the process which has direct access to the local database. For operations which require more complex processing (such as queries or join session operations), it starts manager processes which then process incoming requests.

The GAP server handles the *GUA binding phase*. If a GUA initiates the *user authentication phase* (by issuing a bind user request), the GAP server dynamically starts a GSA user process which is then responsible for handling this user's authentication and, if the authentication has been successful, also the *user bound phase*. The GSA user process terminates if the user unbinds, if the user's authentication failed or if the GUA forces an unbind (ie the GAP connection is terminating). Consequently, for each user bound to the GSA, a GSA user process exists which handles the users requests and delivers any results or notifications to the user.

The GSA manager has the role of the central entity in the GSA design. It accepts all incoming GSP PDUs and processes them according to their content. If a PDU is the result of a operation requested by a GSA user process, it is forwarded to this process. If an incoming GSP PDU requires the GSA to perform some actions, it either performs the appropriate actions itself or it starts a separate manager process which lives as long as the operation is being processed. The GSA manager also has direct access

[14]There were two main reasons for the usage of message queues. First, message queues provide message boundaries for communications between processes, ie it is not necessary to implement some kind of length handling. Second, it is possible to selectively wait for one type or a range of types of messages, which makes it easier to implement synchronization between communicating processes. Message queues do not provide the most efficient interprocess communications mechanism, but since we decided to implement a prototype of a GSA, we accepted the performance implications of using message queues.

[15]Because we use the unicast connection only for sending a response to the requesting GSA, a transactional variant of TCP such as T/TCP described by Braden [8] would be preferable. However, at the moment we use standard TCP, thus including the overhead of the rather expensive TCP connection establishment and the problem with both ends going to the TIME-WAIT state after closing the connection.

to the database, where all local objects and relations are stored. Currently, we use the standard Unix ndbm package for managing the database. This package implements a simple storage for key/data pairs, in our case these pairs consist of an object's or relation's name as the key and ASN.1 BER encoded data.

4.3 Implementation results

The implementation described in the previous sections has been tested in various ways. The majority of tests has been performed as load testing, where a huge load of requests was produced and the behavior of the system has been observed. After these tests (which included a number of test domains with GSAs in the local network and internationally distributed), all timeouts have been adjusted carefully to find the optimal balance between unnecessary repetitions and unnecessary wait periods. Most operations now have five or ten seconds timeouts, assuming that due to the usage of a reliable multicast protocol (which already has internal timers for keep-alive packets), timeouts on GSP level should occur only rarely.

Analyses of the GSA code showed that about 70% of the time spent is used for program logic, 30% is used for database accesses, and only 0.3% are used for coding and decoding ASN.1 data. This was a surprise to us, since the code generated by the Snacc ASN.1 to C/C++ compiler is huge (120000 lines of generated C++ code as opposed to 20000 written lines of C++ code for the various GSA processes). However, this still causes problems, because although the CPU load caused by a GSA running on a system is moderate, due to the size of the processes (most of them including coding/decoding routines), a system running a GSA is heavily loaded by swapping processes from and to memory. More detailed performance figures as well as an evaluation of the implementation are given in [49].

One important point when discussing the performance of a distributed directory service is the scalability of the architecture. GMS can be scaled in three dimensions, which are discussed in the following list.

- *Number of users per GSA.* The current GSA implementation is obviously not suited to support a larger number of users, since every user is represented by a process (as shown in figure 7). However, the resources required for each user could be reduced to a few table entries if the GSA code was designed appropriately. Thus, the number of users per GSA could be fairly big (in the magnitude of a few hundreds) if the GSA implementation was carefully designed.

- *Number of GSAs per domain.* The number of GSAs per domain also is the number of GSAs receiving all multicast requests to this domain and replying to them, if necessary. Therefore, the number of GSAs per domain should be kept fairly small (in the magnitude of ten) to avoid the well-known implosion problem. This imposes no problem, since GSAs are meant to be central services which are remotely accessed using the GMS access protocol (GAP).

- *Number of domains.* The number of domains can be scaled in two ways. Extending the domain hierarchy horizontally does not cause any change in GSP performance, since domain name resolution is not affected by the number of subdomains of a domain. Extending the domain hierarchy vertically (ie introducing new levels of subdomains) influences the domain name resolution linearly, since the domain name resolution requests have to be propagated through more domains. Requests to domains are not influenced at all, since they are directly addressed to the domain. Furthermore, when using caching in the GSA instead of performing a domain name resolution for every request, the effect of extending the domain hierarchy vertically could be minimized. Hence, the number of domains does not influence GMS in a way which could cause performance problems.

Consequently, GMS is able to be used in a large scale[16], provided the number of GSAs per domain is kept reasonably small (which is also preferable from a management point of view). We therefore

[16] In this case, we refer to a large scale (probably world-wide) distribution of session participants and a large number of

believe that the approach to group and session management presented in this paper not only makes group communications platforms more flexible, but also can be used on a global scale.

5 Conclusions

In this paper we have presented the model, architecture, and implementation for a transport-independent group and session management system (GMS). GMS consists of two components, GMS user agents (GUAs), and GMS system agents (GSAs). These components communicate using the GMS access protocol (GAP, for GUA-GSA communication) respectively the GMS system protocol (GSP, for GSA-GSA communication). GMS is designed as a distributed directory service with additional functionality, such as notifications. There is a set of defined object types which may be used to model information about users, groups, sessions, and flows. Mechanisms for authentication and authorization are available and make the design of secure communications platforms possible.

The implementation of the GUA component permits the easy integration into communications platforms. Because the design of object types and operations is independent from a specific transport service, GMS may be used by various communications platforms. The current implementation will be integrated into two platform, the Da CaPo system, and the MCF multipoint communications framework, which both are developed at our laboratory. Future plans include the incorporation of the GUA component into different platforms. It is also planned to extend the GUA and GSA with regard to the transport infrastructures being used.

We believe that GMS in general and the GUA as a component for the inclusion into communications platforms will permit the design and implementation of group communications platforms with a more abstract service in terms of naming, addressing, name and address management, and authentication. We also believe that the implementation of such a system and its usage inside actual group communications platforms will lead us to a better understanding of which services are required from an application point of view and which services should be provided by the GUA component.

References

[1] Paul Albitz and Cricket Liu. *DNS and BIND.* O'Reilly & Associates, Inc., Sebastopol, California, 1992.

[2] 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.

[3] Anindo Banerjea, Domenico Ferrari, Bruce A. Mah, Mark Moran, Dinesh Verma, and Hui Zhang. The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences. *IEEE/ACM Transactions on Networking*, 4(1):1–10, 1996.

[4] Daniel Bauer, Burkhard Stiller, and Bernhard Plattner. An Error-Control Scheme for a Multicast Protocol Based on Round-Trip Time Calculations. In *Proceedings of the 21st IEEE Conference on Local Computer Networks*, pages 212–221, Minneapolis, October 1996. IEEE Computer Society Press.

[5] Daniel Bauer, Erik Wilde, and Bernhard Plattner. Design Considerations for a Multicast Communication Framework. In *Proceedings of the Tenth Annual Workshop on Computer Communications*, Eastsound, Washington, September 1995.

[6] Peter Beadle. Experiments in Multipoint Multimedia Telecommunication. *IEEE MultiMedia*, 2(2):30–40, 1995.

[7] R. Bettati, D. Ferrari, A. Gupta, W. Heffner, W. Howe, M. Moran, Q. Nguyen, and R. Yavatkar. Connection Establishment for Multi-Party Real-Time Communication. In Thomas D. C. Little and Riccardo Gusella,

users, groups, and sessions being managed by GMS, rather than sessions with a large number of associated users. GMS has been designed to support tightly-coupled sessions, for loosely-coupled sessions other approaches (such as the mbone tools) are better suited.

editors, *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, volume 1018 of *Lecture Notes in Computer Science*, pages 240–250, Durham, New Hampshire, April 1995. Springer-Verlag.

[8] R. Braden. Extending TCP for Transactions – Concepts. Internet RFC 1379, November 1992.

[9] R. Braudes and S. Zabele. Requirements for Multicast Protocols. Internet RFC 1458, May 1993.

[10] Andrew Campbell, Geoff Coulson, and David Hutchison. A Multimedia Enhanced Transport Service in a Quality of Service Architecture. In D. Shepherd, G. Blair, G. Coulson, N. Davies, and F. García, 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 124–137, Lancaster, UK, November 1993. Springer-Verlag.

[11] Georg Carle, Jochen Schiller, and Claudia Schmidt. Support for High-Performance Multipoint Multimedia Services. In Hutchison et al. [24], pages 219–240.

[12] S. Chuang, J. Crowcroft, S. Hailes, M. Handley, N. Ismail, D. Lewis, and I. Wakeman. Multimedia Application Requirements for Multicast Communications Services. In Barry Leiner, editor, *Proceedings of International Networking Conference INET'93*, pages BFB-1–BFB-9, San Francisco, August 1993.

[13] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware – Some Issues and Experiences. *Communications of the ACM*, 34(1):38–58, 1991.

[14] Hans Eriksson. MBone: The Multicast Backbone. *Communications of the ACM*, 37(8):54–60, 1994.

[15] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network Support for Multimedia – A Discussion of the Tenet Approach. *Computer Networks and ISDN Systems*, 26:1267–1280, 1994.

[16] M. Handley, J. Crowcroft, and C. Bormann. The Internet Multimedia Conferencing Architecture. Internet Draft, MMUSIC Working Group, February 1996.

[17] Mark Handley and Van Jacobson. SDP: Session Description Protocol. Internet Draft, MMUSIC Working Group, November 1996.

[18] Mark Handley and Eve Schooler. Session Invitation Protocol. Internet Draft, MMUSIC Working Group, February 1996.

[19] Mark Handley, Henning Schulzrinne, and Eve Schooler. Session Initiation Protocol. Internet Draft, MMUSIC Working Group, December 1996.

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

[21] Lutz Henckel. Multipeer Connection-mode Transport Service Definition based on the Group Communication Framework. Technical report, GMD FOKUS, Berlin, June 1994.

[22] Lutz Henckel. Multipeer Transport Services for Multimedia Applications. In S. Fdida, editor, *Proceedings of the IFIP TC6/WG6.4 Fifth International Conference on High Performance Networking*, volume C-26 of *IFIP Transactions*, pages 167–186, Grenoble, France, June 1994. Elsevier.

[23] D. Hutchison, H. Christiansen, G. Coulson, and A. Danthine, editors. *Teleservices and Multimedia Communications – Proceedings of the Second COST 237 Workshop*, volume 1052 of *Lecture Notes in Computer Science*, Copenhagen, Denmark, November 1995. Springer-Verlag.

[24] 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*, Vienna, Austria, November 1994. Springer-Verlag.

[25] International Organization for Standardization. Draft Text on the Subject of "Multi-peer Taxonomy". ISO/IEC JTC1/SC6 N9161/IV, March 1995.

[26] International Organization for Standardization. First Draft of Enhanced Communications Transport Service Definition. ISO/IEC JTC1/SC6, March 1995.

[27] International Telecommunication Union. Data Protocols for Multimedia Conferencing. Draft Recommendation T.120, 1995.

[28] International Telecommunication Union. The Directory – Overview of Concepts, Models and Services. Recommendation X.500, March 1995.

[29] 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.

[30] Laurent Mathy, Guy Leduc, Olivier Bonaventure, and André Danthine. A Group Communication Framework. In Wulf Bauerfeld, Otto Spaniol, and Fiona Williams, editors, *Broadband Islands '94: Connecting with the End-User*, pages 167–178, Hanburg, Germany, June 1994. North-Holland.

[31] Andreas Mauthe, Geoff Coulson, David Hutchison, and Silvester Namuye. Group Support in Multimedia Communications Systems. In Hutchison et al. [23], pages 1–18.

[32] 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.

[33] P. Mockapetris. Domain Names – Concepts and Facilities. Internet RFC 1034, November 1987.

[34] Sassan Pejhan, Alexandros Eleftheriadis, and Dimitris Anastassiou. Distributed Multicast Address Management in the Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1445–1456, 1995.

[35] Thomas Plagemann, Janusz Waclawczyk, and Bernhard Plattner. Management of Configurable Protocols for Multimedia Applications. In B. Furht, editor, *Distributed Multimedia Systems and Applications – Proceedings of the IASTED/ISMM International Conference*, pages 78–81, Honolulu, Hawaii, August 1994.

[36] José F. de Rezende, Andreas Mauthe, David Hutchison, and Serge Fdida. M-Connection Service: A Multicast Service for Distributed Multimedia Applications. In Hutchison et al. [23], pages 38–58.

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

[38] 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.

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

[40] Michael Sample. Snacc 1.1: A High Performance ASN.1 to C/C++ Compiler. Technical report, University of British Columbia, Vancouver, July 1993.

[41] Michael Sample and Gerald Neufeld. Implementing Efficient Encoders and Decoders For Network Data Representations. In *Proceedings of the IEEE INFOCOM '93 Conference on Computer Communications*, pages 1144–1153, San Francisco, 1993. IEEE Computer Society Press.

[42] Henning Schulzrinne. Simple Conference Invitation Protocol. Internet Draft, MMUSIC Working Group, February 1996.

[43] Bohdan Smetaniuk. Distributed Operation of the X.500 directory. *Computer Networks and ISDN Systems*, 21:17–40, 1991.

[44] Ralf Steinmetz and Klara Nahrstedt. *Multimedia: Computing, Communications, and Applications.* Prentice-Hall, Upper Saddle River, New Jersey, 1995.

[45] Clemens Szyperski and Giorgio Ventre. Efficient Support for Multiparty Communication. In Hutchison et al. [24], pages 185–198.

[46] François Toutain and Laurent Toutain. Network Support for Multimedia Communications Using Distributed Media Scaling. In Hutchison et al. [23], pages 139–158.

[47] Erik Wilde. Specification of GMS Access Protocol (GAP) Version 1.0. Technical Report TIK-Report No. 15, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, March 1996.

[48] Erik Wilde. Specification of GMS System Protocol (GSP) Version 1.0. Technical Report TIK-Report No. 19, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, September 1996.

[49] Erik Wilde. *Group and Session Management for Collaborative Applications.* PhD thesis, Swiss Federal Institute of Technology, Zürich, 1997. Diss. ETH No. 12075.

[50] Erik Wilde, Pascal Freiburghaus, Daniel Koller, and Bernhard Plattner. A Group and Session Management System for Distributed Multimedia Applications. In G. Ventre, J. Domingo-Pascual, and A. Danthine, editors, *Multimedia Telecommunications and Applications – Proceedings of the Third COST 237 Workshop*, volume 1185 of *Lecture Notes in Computer Science*, pages 1–22, Barcelona, Spain, November 1996. Springer-Verlag.

[51] Neil Williams and Gordon S. Blair. Distributed multimedia applications: A review. *Computer Communications*, 17(2):119–132, 1994.