IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4480

Reconfiguration in Tightly-Coupled Conferencing Environments

Dirk Trossen, Wolf-Christian Eickhoff
Dept of Computer Science IV, University of Technology Aachen, Ahornstr. 55, D-52074 Aachen
Tel: _49 241 8021414; Fax: +49 241 8888 221; dirk@i4.informatik.rwth-achen.de

ABSTRACT

An efficient conference service facilitates the implementation and run-time control of conference applications. Beside key features of a conference service like conference management, multipoint communication support, and application state synchronization, the reconfiguration of such environments is crucial for the scalability and availability of the service. The paper proposes a reconfiguration mechanism for the Scalable Conferencing Control Service (SCCS) to add the functionality to this service for tightly-coupled conferences. The proposed scheme allows the reconfiguration of running conferences to optimize the system during runtime without blocking the conference during the reconfiguration. The consistency awareness is shown as well as the determination of the costs in terms of time needed for the operation.

1 INTRODUCTION

The communication among distributed applications is enabled by conferencing environments. Progress in networking and computer techniques and increased knowledge of the functionality of conferencing applications are the main pushing factors for *computer supported cooperative work* (CSCW) from a technical point of view. Due to the globalization of companies the increasing costs and distribution of institutions are the pulling factors from an application's point of view because they promise to be an efficient utility for cooperation.

Crucial for conferencing functionality is the *multipoint communication* among several users, facilities for *coordination* and *synchronization* of distributed applications and users, and functionality to inquire for information about the state and structure of existing conferences may be needed. Due to this tight control of the conferencing scenario, *tightly-coupled conferencing environments* are proposed for the realization ([1][10]). Most of these environments are built using tree topologies. Requests, e.g. for *floor control* [2], are routed within this tree. Thus the performance depends on the location of *active entities* within the tree. For optimization of such environments, *reconfiguration* of existing topologies during runtime is crucial to improve the overall performance of the system. But reconfiguration functionality is also needed for simple administration features, e.g. removing a node from the tree when the computer is switched off.

All these features are mostly independent of specific scenarios. Therefore it is useful to provide them by a generic system layer. Usage of the generic functionality accelerates and simplifies the implementation of conference applications. In [10], the *Scalable Conferencing Control Service* (SCCS) is proposed as a generic conferencing service to realize groupware applications with tight-control in terms of floor and access control. In this paper a reconfiguration scheme is proposed to extend the conference management functionality of SCCS to provide e.g. dynamic reconfiguration for topology optimization during runtime of the conference.

The paper is organized as follows. In chapter 2 related work in the area of group communication systems and reconfiguration is presented. In chapter 3 the conferencing service SCCS is introduced, which is extended by the reconfiguration functionality in chapter 4. For this extension the mechanisms, correctness and costs in terms of time used are presented before concluding the paper.

2 RELATED WORK

The ITU specified a set of standards (T.120, [4]) providing basic multipoint and conferencing functionality. A tree-based approach of inter-connected service providers is used to which applications are attached. Re-ordering of providers within the tree is not provided. Due to the centralized approach of the resource management the uppermost provider is the single point of failure within the environment.

The IETF proposed a conferencing control protocol [1], which is very similar to the ITU approach. It offers the same services like the T.120 protocol suite, but there is not yet a protocol specification for this approach. Reconfiguration is not supported due to the similarity to the ITU approach.

Beside special group communication protocols there are platforms for distributed computing in general. The OMG approach CORBA (*Common Object Request Broker Architecture* [7]) provides a standard for the interoperability of distributed objects in heterogeneous environments. Reconfiguration is provided by *migration* of objects ([3][6]) from one server to another which is often used for load balancing, so migration of objects may be used in a set of servers (which may be the providers in a tree-based conferencing environment). But there is often information distributed among the servers which has to be adjusted when changing the logical position within the tree which results in a complex migration and state exchange procedure. Additionally the client is locked during the migration (or to avoid blocking, a migration is canceled when the client is accessing the object during the migration).

A platform for distributed systems was presented by the *Horus* system [8] consisting of a protocol stack architecture of several small protocol entities. The platform was extended for adding group communication and fault tolerance functionality to CORBA [5] by introducing *object groups*.

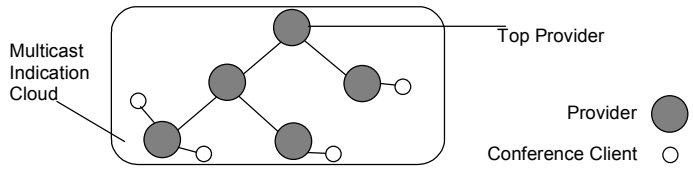
3 SCALABLE CONFERENCING CONTROL SERVICE

This chapter introduces the main features of the *Scalable Conferencing Control Service* (SCCS [10]) to understand the reconfiguration scheme. SCCS provides mainly the following features:

3.1 Conference Management

SCCS is built on top of a reliable multicast transport proto-

Figure 1: Topology of SCCS



col and a reliable unicast transport protocol. SCCS is provided by *providers*, which are organized in a hierarchical tree topology (figure 1) using the reliable unicast transport protocol. Conference clients are connected to one of these providers. Each control request is sent upward in this topology, some indications like database refreshes are sent back via a *multicast indication cloud*, a dedicated multicast channel which is established during setting up the conference. The database of conference members and applications is replicated at each provider in the tree. Thus, read requests are minimized. If a database entry changes, the database is only partially refreshed via the multicast indication cloud. Therefore scalability of this replication is improved. Conferences may be created locally by conference clients which may also join an existing conference at a provider. Existing conferences may also be merged together.

3.2 Multipoint Transfer

Simple point-to-point communication is supported similar to the T.120 by *user identifiers*. For multipoint communication a flexible *channel* concept provides *public* and *private* channels. Channels are only administrated by the providers and directly mapped to multicast groups of the underlying multicast transport protocol, so that routing of multipoint user data is done by the underlying multicast transport protocol to improve the scalability of SCCS.

3.3 Application State Synchronization and Access Control

SCCS provides application state synchronization and access control by a *token* concept similar to the T.120 [4]. Tokens are an abstraction of states or access rights for (real) resources in a group communication environment [2], the mapping of resources or states to tokens is beyond the scope of the protocol. The tokens may be *grabbed* exclusively by one conference member or *inhibited* non-exclusively by several members.

3.4 Resource Allocation

Resources in the conferencing context are *token*s and *channels*. One of the main scalability factors is the response time of requests for such resources. In the T.122 standard a central database is used, which is stored at the top provider to which each request is forwarded, resulting in high response time of requests. In [9] an improved management scheme is proposed providing a high scalability of the tree-based administration of the provider tree, which results in an improved *floor control* [2] of the service. The main idea is, that requests are sent upward in the tree until a provider is reached, who is able to generate a confirm or indication. The providers along the *resource path* only know in which branch of the lower subtree the resource is allocated (figure 2). They do not know the owner of the resource. The resource path is established during the propagation of the allocation confirmation back from the top provider to the originator (figure 2 left). Further requests for the specific resource and location updates may be improved. This results in higher performance for resource requests, especially if the resource requests are local.

Due to the resource-branch mapping there exists routing information which is distributed among the providers along the resource path, which has to be updated during a reconfiguration of the topology as a major task of the reconfiguration.

4 RECONFIGURATION

Conferencing services for tightly-coupled conferences (like T.120 or SCCS) use a tree-based approach to interconnect the service providers. For simple operations like removing a provider from the tree during runtime of the conference, e.g. because the computer is switched off, a reconfiguration scheme is needed. Additionally the performance of the request routing depends on the location of active entities within the tree. Mechanisms like the proposed resource management in SCCS (chapter 3.4) improve the performance of requests which are locally within the tree. For that, reconfiguration of existing tree topologies during the runtime of the conference may be used to improve the performance of a conference after creation of the topology. Basically a *dynamic* reconfiguration should detect the active entities in a conferencing scenario and group them closer together in a reconfigured tree. An approach to detect active entities is proposed in [12] and is not within the scope of this paper. After the determination of the new tree, a *static* reconfiguration mechanism has to build the new topology with the constraints

- to avoid inconsistencies and
- to avoid blocking (parts of) the conference during reconfiguration.

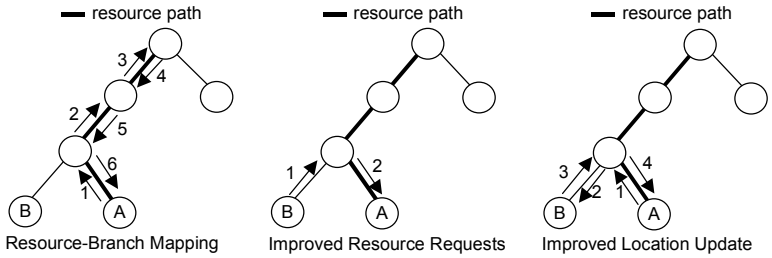
In the following we propose a reconfiguration scheme which fulfils these constraints and offers a complete re-ordering of providers in the tree topology. The proposed reconfiguration provides two operations: *delete* and *move*. Deleting a provider from the tree means to build a new tree without this provider. Moving a subtree (consisting of one or several providers) means to delete the subtree from its original position and to integrate it at its new position in the tree. In the following we restrict ourselves to the description of the reconfiguration algorithm for the move operation to simplify the description of the mechanism, the correctness and the cost determination in terms of time used for the operation.

4.1 Mechanism

This section describes the mechanism of the *move* operation for one subtree. It is very easy to extend the scheme to several reconfiguration operations in one step. The reconfiguration is divided in four steps:

1. Initiation of the reconfiguration

Figure 2: Resource Management in SCCS



2. Establishing the new topology
3. Updating the databases
4. Switching to the new topology

In the following all four steps are explained to clarify the reconfiguration mechanism.

4.1.1 Initiation of the Reconfiguration

The reconfiguration operation may be initiated by any provider in the tree. The operation is forwarded to the top provider, who has the complete information of the tree. The top provider decides whether the operation is valid or not and determines the new topology based on the topology information stored in each provider (which is collected when building the topology). *Reconf_Start_rq* messages are sent to the appropriate providers in the tree. Only providers, who have to start reconfiguration operations, are involved in the reconfiguration operation, thus receive a *Reconf_Start_rq* message. When a provider along the message path receives a message, it determines the new future subtree. If there are changes in its subtree, the provider forwards the message to its lower subtree. The transfer of *Reconf_Start_rq* messages is divided in two steps. First the messages for providers, to which a new subtree is moved, are sent. These messages are confirmed by the appropriate providers. After that, the messages for the providers, who are moving, are sent. The confirmation is delayed until the connection establishment was successful (see next section). Otherwise the message is confirmed with a negative result. In figure 3 *B* has to be moved to *A*. Thus the top provider sends two *Reconf_Start_rq* messages. The first one is send to *A* consisting of the address of *B* and the indication to wait for a move. During the propagation of the message to *A*, the providers along the message path determine the new subtree. *A* confirms the message to the top provider. The second message is sent to *B* consisting of the address of *A* and an indication to move. The new subtree topology is built in all providers along the message paths and in *A*.

4.1.2 Establishing the New Topology

The new topology is built in parallel to the old one to avoid blocking the conference. When a provider receives a *Reconf_Start_rq* message with a move indication, it establishs a connection to the superior provider given in the message. After the connection establishment the moved provider sends a *Reconf_Start_cf* to the top provider via the new connection to indicate the completion of the connection establishment. In figure 4, *B* establishes a connection to *A* and sends a *Reconf_Start_cf* to the top provider using the new connection. These messages are collected in each provider when there are several outstanding messages in the subtree of the provider, generating a cumulative confirmation. When the top provider received all outstanding messages the new topology establishment is finished and the update of the databases begins. During this phase all requests are sent using the old topology, the new one is only prepared for reconfiguration, but it is not yet used!

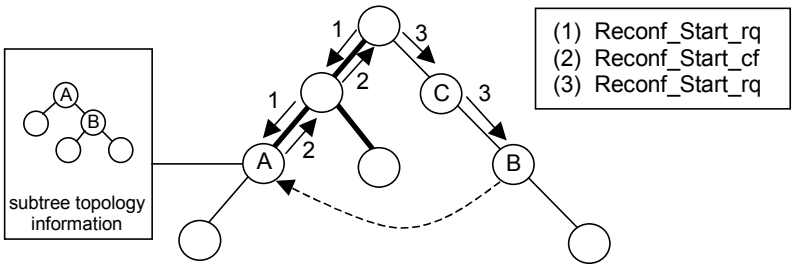
4.1.3 Updating the Databases

In SCCS there are three different databases [11], which have to be updated due to the reconfiguration:

1. Topology database
2. Conference database
3. Resource database

The first one is distributed among each provider containing information of the subtree below the given provider. This information is updated implicitly during the propagation of the *Reconf_Start_rq* messages. Figure 3 shows the topology database of provider *A* and *C*.

Figure 3: Starting the Reconfiguration



The second database is updated only when a delete operation is invoked. For that the multicast indication cloud is used. A move operation does not need to update this database, because there is no change in any member information.

The third database is the most important one, because the resource paths of the resource allocation scheme (chapter 3.4) are stored in this database and this information is distributed among the providers. Due to the reconfiguration the resource paths may change and have to be updated.

After the establishment of the new topology, the top provider sends a *Reconf_Update_rq* message in these subtrees, which are reconfigured with the indication to update the resource database, using the old topology. The providers along the path of that message switches the future routing of resource requests to the new subtree, when the message has passed. The providers with a new superior node register the resources in their *new* subtree *without* an outstanding request by sending the resource information upward using the new connection, until a provider is reached, who is able to confirm the new registration (the registration cannot be denied because the resource path is only re-mapped but the resource has been already allocated). When there is an outstanding resource request the information update for that resource is forwarded after the completion of the operation. After the first forwarding of resource information ALL further resource requests in the moved subtree are sent using the new topology for consistency with the forwarded resource information. The confirmation for the registration is sent back to the provider with a new superior node. After receiving all confirmations for all updated resources, this provider sends a *Reconf_Delete_rq* via the old topology containing the resources of the new subtree to mark the old resource paths for deletion (when there are no resources to delete a dummy message is sent upward). When there is a provider along the old path with newer information about that resource or another resource-branch mapping (non-exclusive resource) the provider deletes the

Figure 4: Building the New Topology

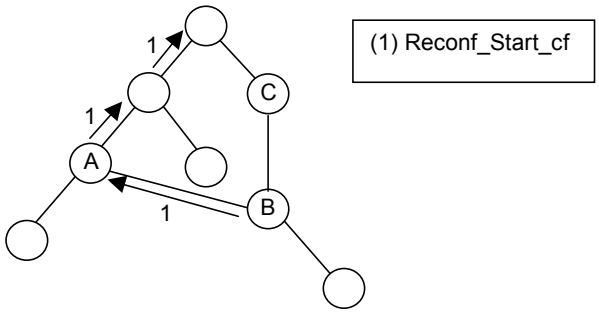
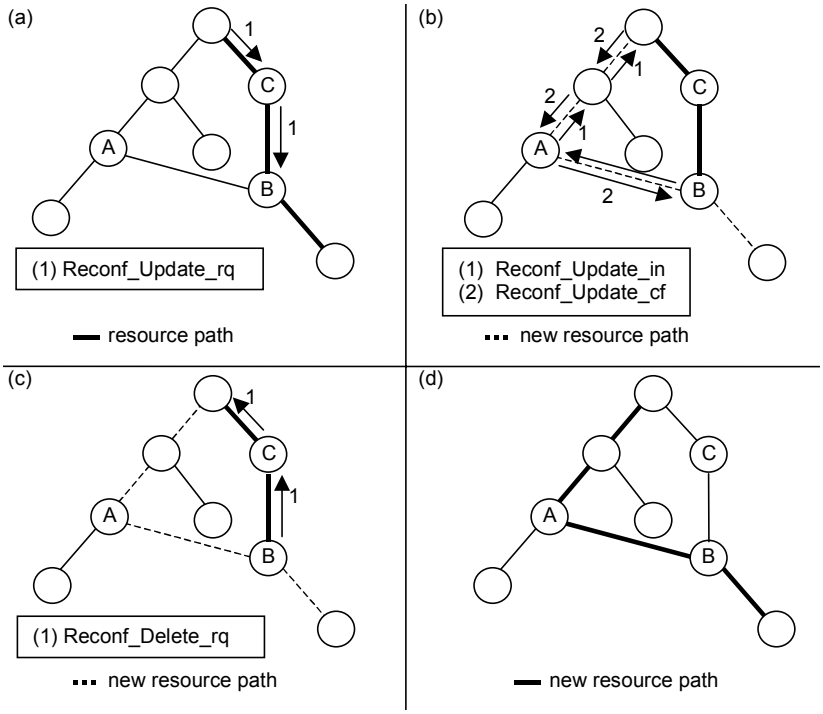


Figure 5: Updating the Resource Path



corresponding resource from the message. The deletion (or the dummy) message is forwarded to the top provider.

In figure 5 the top provider sends a *Reconf_Update_rq* message to *B* who has moved to *A* as its new superior node (figure 5a). *B* sends its entire resource information along the new connection to register the resource in its subtree up to the top provider, because it is not yet allocated in the left subtree. Thus a new resource path for the resource in the moved subtree is established (figure 5b). The top provider confirms the resource update and *B* sends a *Reconf_Delete_rq* message via the old connection (figure 5c). Thus a new resource-branch mapping exists (figure 5d) in the new topology, which is directly used after forwarding the resource information using the new topology.

4.1.4 Switching to the New Topology

When all outstanding deletion messages for resources (including the dummy message when there are no resources in a moved subtree) are received at the top provider, a *Reconf_Delete_cf* message is sent downward via the new route with the indication to delete the old connection at the moved provider (B in our example). The connection is disconnected, when there are no outstanding messages on that link. The disconnection message is confirmed to the top provider using the new routes and the reconfiguration is completed as shown in figure 6.

4.2 Correctness

During updating the databases, requests may proceed to avoid blocking the conference without leading to failures due to inconsistent identifiers or database entries. The topology database is updated implicitly during the propagation of the *Reconf_Start_rq* messages and is enabled after the propagation of the *Reconf_Delete_cf*. The conference database is updated similar to a normal delete or add operation in SCCS by using the multicast indication cloud.

For the consistency of resource requests the correctness of the reconfiguration scheme is shown only for token operations, which may occur during the reconfiguration to simplify the proof. These operations are: *allocation, release, pass, ask* and *inquiry for owners*. The specification for these operations (defined in terms of *Message Sequence Charts* in [11]) is used for the following correctness proofs:

Lemma 1: token allocation, release and pass do not lead to inconsistency

Only resources without pending operations are updated by the moved provider. Resource(s) with pending operations are updated in further requests until all resources in the subtree are updated and consistent. After the first update request the new subtree is used, so that resource requests for new resources which occur after the first update request are consistent.

Lemma 2: token ask operations do not lead to inconsistency

There are two cases to consider. In the first one the token ask operation is forwarded upward and reaches a provider with a *marked for deletion* resource path for that resource. The provider forwards the request upward until it reaches a provider with new (valid) resource information from the updated resource path. That path is established at that moment, because the mark for deletion is done after the resource update. Thus the token owner is reached.

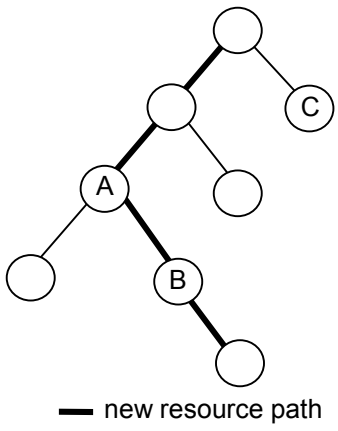
In the second case, the token ask request is already forwarded in downward direction (because it reached a provider with still valid resource information on the old resource path). In that case the old resource path is used until the resource owner is reached.

Lemma 3: Inquiring for owners operations do not lead to inconsistency

The inquiry request is routed to the (old) top provider for coordination of the operation. The top provider uses the old resource paths. Similar to lemma 2 the request is forwarded downward using the old topology regardless of the deletion mark. If the moved provider receives a *Reconf_Delete_cf* message, the disconnect operation is suspended until any pending inquiry operations in the moved subtree are confirmed and forwarded to the old superior node. Thus valid token owner information of the moved subtree is inquired.

With lemma 1 to 3 it is shown that the proposed scheme

Figure 6: New Topology with New Resource-Branch Mapping



provides a non-blocking reconfiguration mechanism with ensured consistency.

4.3 Costs

In this section the costs of a *move* operation is determined in terms of time to complete the operation. For simplification it is assumed that just one subtree is moved and that the top provider remains the same. Additionally it is assumed that the transmission time t_t on all links and the service time t_s in all providers are the same. Additionally t_{CO} is defined as the time to connect a moved provider to a new upper provider, t_{DIS} as the time for a disconnect operation, l_m as the level of the moved and l_n as the level of the new superior provider in the old tree.

The costs determination is divided in the same four steps, which were used for the reconfiguration mechanism. For the first step two messages are sent to the moved and the new superior node sequentially. The new superior node confirms the message. Thus the initialization costs for that step are given by

$$t_{init} = 2 \cdot t_t + 2 \cdot t_s \cdot (l_m + l_n + 1) \quad (1)$$

In the second step, the moved provider establishs a new connection to the new superior node and confirms the establishment to the top provider using the new connection. Thus the connection establishment costs are given by

$$t_{connect} = t_{CO} + (l_n + 1) \cdot t_t + l_n \cdot t_s \quad (2)$$

For the third step it is assumed that there are no outstanding operations on any resources in the moved subtree and that, in the worst case, the update has to be forwarded to the top provider. Therefore the resource update is forwarded in one step after receiving a *Reconfig_Update_rq* with a confirmation from the top provider and the deletion of the resources along the old path. Thus the costs are given by

$$t_{update} = 2 \cdot (l_m + 1) \cdot t_t + 2 \cdot (l_n + 1) \cdot t_t + l_n \cdot t_s \quad (3)$$

In the last step, the moved provider is requested to disconnect from the old superior node and to confirm the operation, which leads to the following costs

$$t_{switch} = 2 \cdot (l_m + 1) \cdot t_t + l_m \cdot t_s + t_{DIS} \quad (4)$$

The total costs of the move operation are the sum from (1) to (4):

$$t_{reconfig} = t_{CO} + t_{DIS} + t_s \cdot (3 \cdot l_m + 7 \cdot l_n + 5) + t_t \cdot (3 \cdot l_m + 7 \cdot l_n - 5) \quad (5)$$

Typically t_{CO} and t_{DIS} are much larger (several hundreds of milliseconds) than t_s (a few milliseconds) and t_t (depending on the link speed). When moving several subtrees most operations (e.g. the connect and disconnect operations) are invoked in parallel or are sent in combined messages (e.g. the initialization of the reconfiguration), so that it is expected that the costs are not much higher when moving several subtrees.

5 CONCLUSIONS

This paper introduced a reconfiguration scheme for the *scalable conferencing control service* (SCCS) which was proposed as a generic system layer for tightly-coupled conferences. The reconfiguration scheme provides complete re-ordering and deletion of selected providers within the tree topology of SCCS during runtime of the conference with avoidance of inconsistency and blocking the conference. The correctness of the scheme in terms of consistency was shown and the costs for the reconfiguration were determined in terms of time needed to complete the reconfiguration.

To sum it up it can be said, that the proposed mechanism provides reconfiguration in a scalable conferencing environment for tightly-coupled conferences in contrast to existing conferencing environments. Together with the improved resource management scheme of SCCS scalability can be improved by detecting active entities in a conference scenario and grouping them dynamically during runtime of the conference, which is proposed in [12].

Further work will concentrate on the implementation and integration of the reconfiguration in SCCS. The mechanisms will be tested both on implementation and simulation level. Together with the dynamic reconfiguration, the improvement of the scalability in terms of response time will be shown.

6 REFRENCES

[1] C. Borman, J. Ott, C. Reichert: *Simple Conference Control Protocol*, ftp://ftp.ietf.org/internet-drafts/draft-ietf-mmusic-sccp-00.txt, 1996

[2] H.-P. Dommel, J.J. Garcia-Luna-Aceves: *Floor Control for Activity Coordination in Networked Multimedia Applications*, Proc. of 2nd Asian-Pacific Conference on Communications, 1995

[3] A. Goscinski (1991): *Distributed operating systems: the logical design*, Addison-Wesley, New York, 1991

[4] ITU-T Study Group 8, ITU-T Recommendation T.120: *Data Protocols for Multimedia Conferencing,* 1995

[5] S. Maffeis: *Adding Group Communication and Fault Tolerance to CORBA*, Proc. of USENIX Conference of Object Oriented Technologies, 1995

[6] M. Nuttal: *Survey of Systems Providing Process or Object Migration*, Imperial College Research Report DoC 94/10, Department of Computing, Imperial College, London, 1996

[7] OMG: *The Common Object Request Broker: Architecture and Specification*, 1993

[8] R. Renesse, K. Birman, S. Maffeis: *Horus: A flexible Group Communication System*, Communication of the ACM, 1996

[9] D. Trossen, P. Papathemelis, T. Helbig: *Improved Resource Management for the ITU T.122 Standard*, Proc. of IEEE Workshop on Systems Management, 1998

[10] D. Trossen, K.-H. Scharer: *SCCS: Scalable Conferencing Control Service*, Proc. of 7th IEEE International Conference on Computer Communication and Networks, 1998

[11] D. Trossen: *Scalable Conferencing Control Service - Protocol Specification*, Technical Report 001-99, University of Technology Aachen, 1999

[12] D. Trossen, P. Kliem: *Dynamic reconfiguration in tightly-coupled conference environments*, Proc. of SPIE International Symposium Voice, Video, & Data Communications, Conference on Multimedia Systems and Applications II, 1999