11

DiSC - An Object Oriented Distributed Session Control Implemented with Java

E. Wedlund, C. Johnsson***
Ericsson Telecom AB
Switchlab, Dialoggatan 1, S-126 25 Stockholm Sweden
Phone: +46 8 719 85 96, +46 8 719 4903***
Fax: +46 8 719 66 77
etxelin@kk.ericsson.se, qtxcjo@kk.ericsson.se***

B. J. Olsson
Communicator Teleplan AB
Box 1310, S-171 25 Solna, Sweden
Phone: +46 8 7644500
Fax: +46 8 7644066
bengt.j.olsson@communicator.se

Abstract

Traditional session control is not a sufficient solution in a multimedia services network, it must be renewed in order to handle multimedia and multiuser sessions. In this paper we address this issue and present a proposal for a solution, which we call Distributed Session Control (DiSC). DiSC can be seen both as an overall solution for session control in a network, and simply as a tool for creating complex multimedia sessions. One of the strengths of DiSC is that it does not dictate a new type of network architecture, but will operate on those already existing.

Keywords

signalling, session control, network management, service provisioning, middleware

Intelligent Networks and Intelligence in Networks D. Gaiti (Ed.)
Published by Chapman & Hall

1 INTRODUCTION

This paper describes an object oriented Distributed Session Control (DiSC) for multimedia services networks. The demands placed on a multimedia service network are that it should be flexible, in terms of introducing new services, and that it possibly should support QoS and security, and thereby also billing. Introducing new services can be simplified by moving session and network related issues to a dedicated session control function, such as DiSC, instead of having to deal with it in the applications. Services in the network are expected to be distributed, i.e., a service application can be located so that it is accessible by several/all users in the network. This is supported by e.g. Java and CORBA.

The difference between a multimedia services network and an "old" network, regarding the session control, is that before, a session mainly consisted of two users and one service (POTS). In a multimedia services network, a session can consist of an unlimited number of users and services. In order to support this, a new type of session control is needed. The requirements that can be placed on this session control are:

- Complexity must not grow more than linearly with the number of users and services in a session.
- A user, or an application started by the user, should be able to use an appropriate QoS (if supported by the network).
- It should be possible to charge a user based on different session characteristics.
- It should be possible to use different networks for the same service.

Traditional telecommunication session control implies too much complexity to handle multiuser, multimedia sessions. DiSC solves the complexity problem by using an object oriented view of the session. Distribution of the session control also reduces complexity as well as scaling problems, since each user only keeps information that is relevant for that user.

DiSC is more light-weight than other approaches for multimedia services networks, e.g. TINA (Chapman et al. 1995), in that it only provides a solution for the session handling of the network, which means that it can be used on any existing network. DiSC is not defined for a specific network, which is the case for e.g. the ITU-T H.323 series. In fact, DiSC can interwork with these protocols for creation of connections on various networks, see figure 1 for an example with the H.323 series.

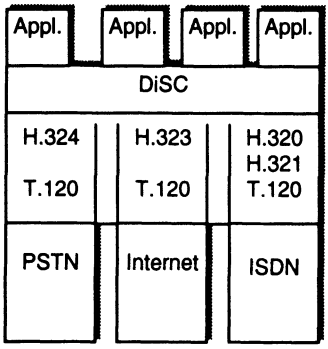

Figure 1. Network Integration through DiSC interworking with the H.323 series.

2 DISTRIBUTED SESSION CONTROL

A project at Bellcore, called the EXPANSE project (Minzer 1991, 1993) introduced an object oriented call model, and a concept of each user having a User Request Manager (URM), which handled negotiation for session establishment. This project was focused on the ISDN, and we have not seen anything from it in the last years. Since we found it to be a useful way of handling the session control, not only for ISDN, we decided to use it in our project.

The Distributed Session Control (DiSC) uses the same call model as the one in the EXPANSE project at Bellcore (Minzer, 1991, 1993). In DiSC, each user also has its own User Request Manager (URM), which handles all negotiation with other URMs for creating and tearing down sessions. The URM can either be placed in the user's computer or in another place in the network.

When discussing DiSC, the term "user" does not necessarily mean a human being, but can also be a distributed application compliant with DiSC, e.g. a media on demand server, or a video conferencing service. This means that there are two ways of using DiSC - as a person to person communication and negotiation, where an application is used for transferring and presenting data between users, and as a tool for a person to utilise distributed applications.

DiSC offers a very flexible and generic model of session control, but one example of its use in a commercial environment is how DiSC can be used by a network operator, which is shown in Figure 2. The service provisioning can be centralised and administered by the network operator, who would benefit from the higher service content. The user would also benefit, since he will be offered compatible services and an easier way to utilise the network(s). In Figure 2, the application server stores applications that can be downloaded when necessary instead of residing at the user's computer. The context server keeps information of

what users are possible to contact, and the URM server can have many user's URMs running.

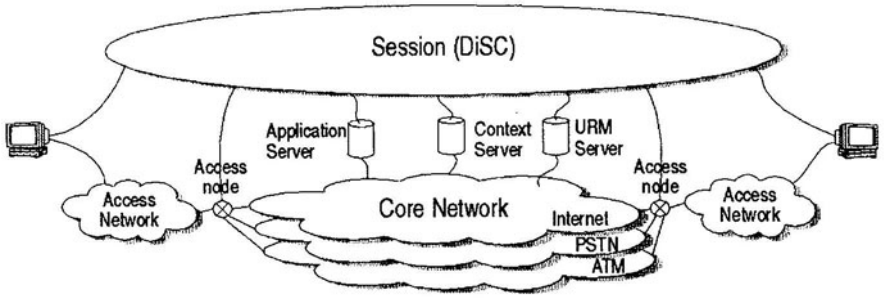

Figure 2.

2.1 A Brief Description of DiSC

Figure 3 shows a network scenario with a user "Claes" who has his URM in his own computer, and a user "Elin" who has her URM in another computer in the network. They are involved in a video conference, where the audio part uses a separate connection. How this session was established will be described further below.

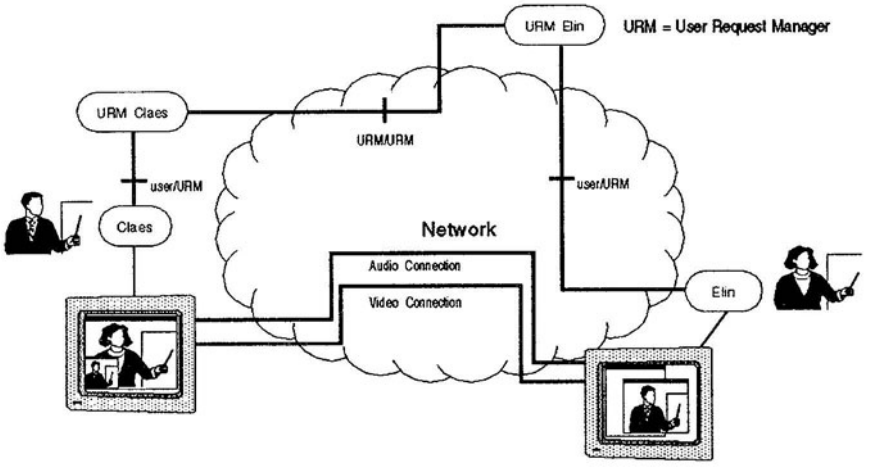

Figure 3. A two-party video conference session.

The User Request Manager (URM) has a LocalView, which describes the session as a tree of objects - DiSCObjects (in EXPANSE denoted call objects). The LocalView only describes the session as viewed from the user, and contains only

information that is relevant for the user. Each DiSCObject in the LocalView represents an essential part of the session, e.g. what user is involved, what service is being used, and so on. The signalling for creating, modifying, or tearing down a session is represented as methods for creating or deleting DiSCObjects in the LocalView. The objects in the LocalView, and their relations, are presented in Figure 4, and listed in Table 1.

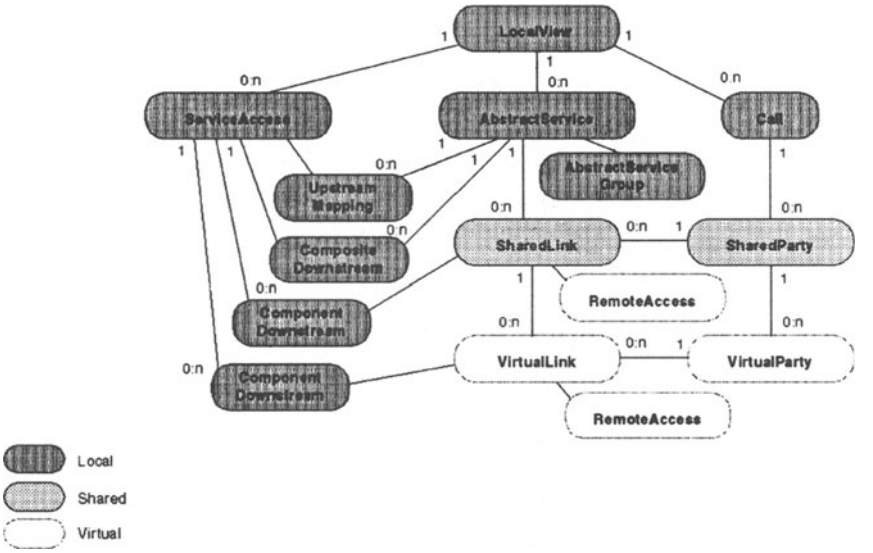

Figure 4. The LocalView.

Table 1.

Object	Explanation
SharedParty	An object representation of a user. Creation of a SharedParty requires confirmation from the remote user which the SharedParty represents. When a SharedParty is created, a corresponding SharedParty is created in the LocalView of the remote user. The SharedParty may contain VirtualParties (see below).
VirtualParty	A reference to a SharedParty in another LocalView.
Call	A call is an association of SharedParties.

SharedLink	Associates a SharedParty with an AbstractService, i.e. defines what service(s) the SharedParty will use for communication.
VirtualLink	A reference to a SharedLink in another LocalView. The user is free to "use" the VirtualLink by creating mapping elements to it. A user can not affect the creation or deletion of a VirtualLink.
AbstractService	Defines a communication service in abstract terms, e.g. video, text, data, etc.
ServiceAccess	An object representation of the network access for the given service. Attributes in the ServiceAccess can be what protocol to be used, if the channel is uni- or bi-directional, a port number etc.
RemoteAccess	An object that describes the ServiceAccess and Mapping at the remote user.
AbstractServiceGroup	AbstractServices may be grouped together in order to enable e.g. synchronisation between different services.
UpstreamMapping	Associates a ServiceAccess with an AbstractService, and makes data produced by the user accessible to the parties in the Call.
CompositeDownstream	A CompositeDownstream mapping element takes data from all the parties associated with that AbstractService and presents it to the user.
ComponentDownstream	The ComponentDownstream mapping element takes data from a specific link (shared or virtual) and presents it to the user.

The DiSCObjects can be divided into three groups: shared, virtual, and local DiSCObjects:

- Shared DiSCObjects can not be created without consent from the remote side. They are always created in pair, e.g., if the user Elin wants to create a SharedParty Claes in her LocalView, she must ask the user Claes to create a

SharedParty Elin in his LocalView. If Claes agrees to that, both objects will be created.

- Virtual DiSCObjects are references to shared DiSCObjects in other LocalViews. For instance, let us say that Elin and Claes have agreed to create the SharedParties as stated above. If Elin later on decides to call another user Bengt in the same call, she will inform Claes of this by telling him to create a VirtualParty hanging under the SharedParty Elin in his LocalView. She will also tell Bengt that there already is one person involved in the call by telling him to create the VirtualParty Claes. A user can not manipulate a virtual object, but it is possible to e.g. create mapping elements to a VirtualLink. The purpose of having shared and virtual objects is to point out who is responsible for that object. Virtual objects are also used for supplying information.
- Local DiSCObjects are objects that only the user has control of. The user may create or delete local objects, although deletion of a local object may cause deletion of shared object, which requires negotiation with other users. For instance, a call is a local object, but in order to delete a call, all SharedParties in it must be deleted. Vice versa - if all the parties in a call have been deleted, there is no use in keeping the call, so it is automatically deleted.

The LocalView in "URM Claes" from Figure 3 is presented in Figure 5. The negotiation needed to create it is presented in Figure 6.

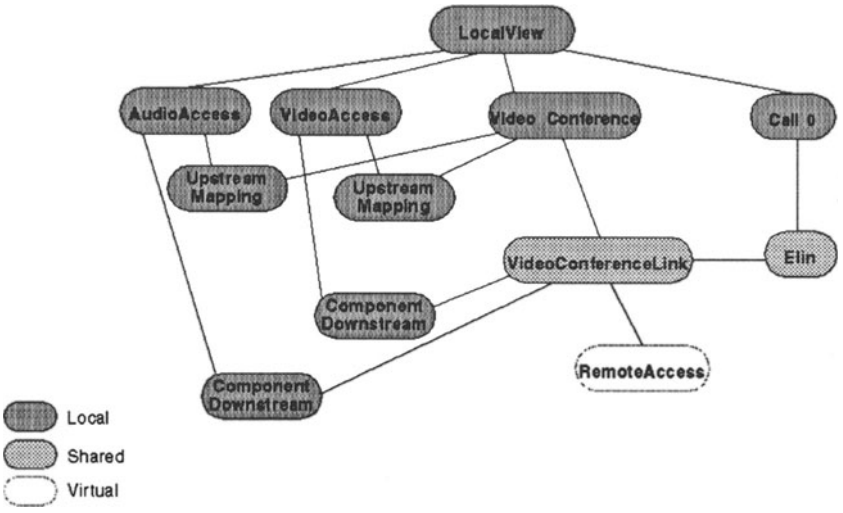

Figure 5.The LocalView for URM Claes.

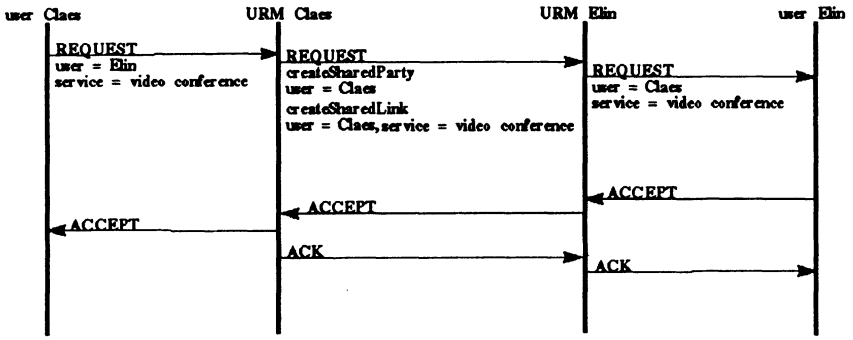

Figure 6. A request.

What is not shown in Figure 6, is the creation of the local objects. Some of them, like a Call or an AbstractService are created automatically by the URM. The mapping can also be created automatically, but the user should be able to change the mapping according to his needs by sending a request to his URM. This is further discussed in section 4.

3 DISC INTERFACE - JAVA VS. CORBA

We have implemented DiSC in Java, and the communication between the URMs is carried out by making Remote Method Invocations (RMIs). This simplifies the implementation of the user/URM, and URM/URM interfaces, but binds DiSC to be implemented in Java. A possibility is to use CORBA for the RMIs, which will allow for any language to be used for the implementation of the user and/or URM. The drawback is that the interfaces will become more complicated to implement. The differences between Java RMI and CORBA are listed in Table 2.

Table 2 Java vs. CORBA

subject	CORBA	Java RMI
Object Types	The code for an object that is passed in a method invocation must exist on both sides of the call. Therefore, objects are passed as the type declared by the interface, even if they are of a derived type.	If the code for an object that is passed in a method invocation does not exist, it is downloaded. Therefore, the object type is preserved, even if the interface has declared a more general type.

Complexity	The objects in the implementation must be mapped onto the object model of the Interface Description Language (IDL) in CORBA.	Since the RMI interface is written in the same language as the rest of the code, this does not imply any difficulties.
Garbage Collection	CORBA supports programming languages that do not have garbage collection. Therefore there is no garbage collector, and the programmer must know when an object should be disposed.	Java has a garbage collector which automatically removes an object if it has no references to it.
Security	No support for security specified.	Support for security.

Another technique for object distribution is Microsoft's DCOM, which like CORBA, is language independent. It has garbage collection, support for security, and can use any available network protocol (not just TCP/IP). However, it has only been available for Windows NT until now, which is one of the reasons why we have not used it.

Our experiences from using Java and Java RMI, are that the RMIs takes a large amount of time, even though this is improving continuously. We also regret that there is no possibility to control the number of TCP connections that are created to carry the RMIs, and that you may not use UDP for this, which probably would improve performance.

We have decided to continue to use Java RMI, mainly to avoid the work implementing a CORBA interface would require. We also anticipate that if the development of Java and Java RMI will continue at the same speed as it is now, it will be a strong competitor to CORBA.

4 ISSUES FOR DISCUSSION

Currently, we have implemented DiSC as a simple conference tool with a few services available on an IP network - since our initial objective simply was to try out how DiSC could be implemented. We have experienced that it is difficult to keep the flexibility of the LocalView when implementing it, since different services can have completely different attributes, and allow different types of session configurations. We have also come to the conclusion that the most likely

future for DiSC is to be used as a middleware between an application and the network, and not as a tool itself, and therefore we intend to define an Application Programmer Interface (API). These issues are discussed in the sections below.

4.1 Rules

The LocalView can describe practically all possible and impossible connection scenarios. It is therefore necessary to apply rules to DiSC so that it only describes sessions that are possible to actually create in the real world. For instance, it is theoretically possible to describe a session with one user transmitting voice, and the other user converting it into a video stream before presenting it on the screen. In most cases, we do not want the URM to allow this, since there probably are not many users who have access to a voice to video converter. However, since we do not want to limit the model, the rules should be able to change, both in time and depending on the environment.

Other situations where rules should be applied is e.g.: Should mapping be created automatically when a SharedLink or AbstractService is created? When are network resources reserved? Who is responsible and thus accountable for the call? In these cases, the rules can depend on the type of service, and a user should be able to make personal rules.

The best solution is probably to have a database with rules from which the URM can check what it is supposed to do. The user should be able to update the database with personal rules on e.g. creation of mapping objects, and the network administrator can update it with new network devices etc.

4.2 Network and Application APIs

DiSC introduces an abstract layer of associations between users and services. In order to create the necessary connections, the LocalView must be instantiated in the network. Also allocation of resources for synchronisation, data conversion etc. might need to be done. DiSC currently uses existing APIs to handle e.g. sockets.

As stated before, there are two ways of using DiSC from an applications point of view. From DiSC's point of view, there are three types of applications, which require two different types of interfaces:

- Traditional applications. Already existing applications that handle the creation of connections themselves, for instance through WinSock in a Windows environment. What DiSC can offer to these applications of course depends on how each application sets up connections. DiSC cannot directly interfere with the creation of connections, but in some cases, it might be possible to improve it through DiSC, e.g. by letting DiSC override the WinSock if that is what the application uses (see Figure 7). What DiSC can do here is to know if the

connection should be multicast or singlecast, and to let the user decide on some attributes of the connections.

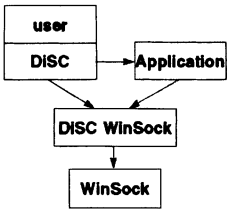

Figure 7 Traditional Application and DiSC

- Collaborative applications. Applications that are used for user to user or group communication, but which expect that the session control and connection creation will be handled by DiSC, see Figure 8. This type of application will replace the DiSC GUI, and use a DiSC API instead of a network interface. DiSC will handle the creation of connections, so that the application does not have to care if the connection is a multi- or singlecast, if the data must be converted etc. Primitives in the DiSC API would be such as "add user", "create mapping", "change attribute", etc. We are currently working on defining an API.

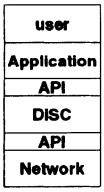

Figure 8 Collaborative Applications and DiSC

- Server applications. Applications that can act as an end user in DiSC. This can be a media on demand server, which users can call through DiSC. Such applications can be written using the DiSC API, so that they can send and receive requests, answer requests etc. (see Figure 9). For instance, if a user calls a service, the service knows what capacity it needs and can respond to the request by demanding the necessary bandwidth.

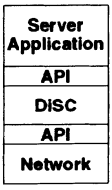

Figure 9 Server Applications and DiSC

4.2.1 Creation of Connections

There are mainly two issues that imposes some trouble regarding the creation of connections, and both are introduced by the distribution of the session control. The first one is how to decide whether to create a multicast or unicast connection. If a connection is created when a downstream mapping element is inserted in the LocalView, it is possible to check if the other user(s) have created mapping elements, since this information is kept in the RemoteAccess object. But if none of the others have created downstream mapping elements, there is no possibility to know if the other users in the session are going to create a downstream mapping element. If they will, it would be a good idea to create a multicast connection. This can depend on the type of service being used, e.g. in a video conference, you might not want to see every person you are talking to.

The other problem is: When a session has been established - which URM should create the connection? This also addresses the rule issue: One could have a rule that says that it is the URM who initiated the session who should create the connections. This is not sufficient, though, since a session can be established without anyone wanting to transmit anything yet.

In EXPANSE (Minzer 1993), the network interface is implemented as two levels under the session control level. The first level, called the Transport Resource Manager (TRM) gets a LocalView that has been transformed so that it is easier to identify the network components needed. The URMs communicate so that they do not do overlapping work, and that they can recognise when a connection should be uni- or multicast.

It seems though that it should be possible to make this simpler by always creating mapping elements as soon as a SharedLink is created. This would on the other hand reduce the flexibility of the creation of connections, but again, this can be set by the rules so that a user may choose if he wants a slower but flexible connection setup, or a fast but forcing one.

4.3 Scaling and Complexity

Since DiSC is distributed, scaling problems will not arise just by introducing DiSC to a large network. Perhaps if the URMs are not placed at the users, they might require a lot of memory at the "URM server". This can be solved to some extent by creating semi-centralised URMs that can handle a limited number of users.

If DiSC is to be used for services where a large number of users can be involved in a session, a considerate number of objects will be created and stored. The VirtualParties and VirtualLinks will not use much space, though, since they only are references to SharedParties and SharedLinks in another LocalView. So the one who will take the heavy part is the one with the most SharedParties and SharedLinks, which probably is a part that is responsible for the service somehow. The users (clients) will not have to store large amounts of perhaps irrelevant data.

5 CONCLUSIONS

A key feature of DiSC is that it hides the complexity of the session control from the user and application, which simplifies introduction of new services in a network. An application can be written for DiSC using an API, but already existing applications might also be used with DiSC. Another benefit is that DiSC itself does not require changes on lower levels in the network, since connections can be handled in a traditional way once the session has been created (separation of call and connection control). DiSC supports charging, since it has information about the session on a high level, which can be logged and used for e.g. service based charging.

The main issue when implementing DiSC is how to get the most out of the potential given in the model, without constraining it. As stated in section 4.1, dynamic rules are necessary for customising DiSC, still keeping the flexibility of the model. It is also important, when writing the DiSC API, to make the primitives as general as possible.

Regarding the question if RMI, CORBA, or DCOM should be used, it seems to be a good idea to wait a while before deciding, since RMI still is quite new and unexplored. We will continue our research work using Java for simplicity reasons, but if a commercial system should be created, CORBA might - for the moment - be the best solution.

6 REFERENCES

Chapman, M. and Montesi, S. (1995) Overall Concepts and Principles of TINA, Version 1.0.

Minzer, S. (1991) A Signalling Protocol for Complex Multimedia Services. IEEE Journal on Selected Areas in Communications, Vol. 9, No. 9, 1383-94.

Minzer, S. (1993) EXPANSE Software for Distributed Call and Connection Control. International Journal of Communication Systems, 7, 149-60.

Object Management Group (1997) The Common Object Request Broker: Architecture and Specification.

Weiss, M., Johnsson, A., and Kiniry, J. (1996) Distributed Computing: Java, Corba, and DCE. Open Software Foundation Research Institute.

Sun Microsystems Inc. (1997) Java Remote Method Invocation Specification.

7 BIOGRAPHY

Elin Wedlund received the M. Sc. degree in Electrical Engineering from Lund Institute of Technology in Sweden in 1996. Her master thesis was on the subject of TCP and the ABR service in ATM. Since her graduation she has been working

at SwitchLab, which is a lab within Ericsson's applied research organisation, where she studies networking issues.

Claes Jonsson started working at Ericsson in 1995, and has worked on the development of AXE switches before he came to SwitchLab. Alongside his work at Ericsson, he is also a student at the Royal Institute of Technology in Stockholm, where he studies Computer Science. At SwitchLab, he works with implementation issues.

Bengt J. Olsson received the M. Sc. degree in Engineering Physics 1985 and a Ph. D. in Atomic & Molecular Physics 1989 from the Royal Institute of Technology in Stockholm, Sweden, and did his Post doc. at the University of Wisconsin in 1990. He started working at Ericsson in 1991 with system design on SDH systems, and moved to Ellemtel Telecommunication Systems Laboratories in 1994 to become leader of the Network Studies group at SwitchLab. Since April 1997 B.J.O. works as a data/telecom consultant at Communicator Teleplan AB of Stockholm.