Floor Control for Multiparty Sessions in 3G Networks

Samer Hawwa

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

January 2004

Your file Votre référence
ISBN: 0-612-91039-3
Our file Notre référence
ISBN: 0-612-91039-3

ABSTRACT

Floor Control for Multiparty Sessions in 3G Networks

Samer Hawwa

Value-added services in telecommunication networks may be defined as anything that goes beyond two-party voice calls. Today's mobile networks primarily use circuit switched networks to deliver voice and data. Tomorrow's networks, third generation (3G) networks, are built on the foundation of IP technology capable of delivering multimedia services over packet switched networks.

This thesis focuses on the floor control issue for multiparty sessions in tomorrow's telecommunication networks. Floor Control enables the control of media streams, e.g., who can be seen and heard when more than one party is exchanging media streams over a network.

Several conferencing scenarios have been studied, leading to a definition of the requirements for a generic floor control mechanism. After defining the requirements, we surveyed the related work in the literature (i.e., SIP and PARLAY) to identify whether they provide (or not) the defined requirements. Based on this evaluation, we have proposed an architecture that benefits from the available technologies and which also provides the missing parts in order to achieve a complete solution.

We have identified that PARLAY as per today's specification does not fulfill all the requirements for floor control. In addition, we have identified the need for a complete solution for floor control integrating APIs with the protocols.

We propose new extensions for floor control to the PARLAY APIs. We identify the main components for floor control and their role in a multiparty session. We describe how these components interact with each other. We also propose an overall architecture for floor control combining protocols with APIs.

We explored emerging technologies (i.e., Web Services) as one step further in our study. We learned that the use of Web Services is suitable for floor control in multiparty sessions applications development. A real Web service development environment (i.e., UDDI, WSDL, and SOAP) could be offered. It provides a higher level of abstraction than the PARLAY API, hiding the complexity of the telecommunication networks. It offers more flexibility with application-to-application communication, and faster time for development and deployment.

We have built a video conferencing application as a proof of concept to demonstrate the new interface for floor control as a Web Service, the proposed extensions for floor control in the PARLAY APIs and the overall architecture combining the protocols with APIs.

ACKNOWLEDGMENTS

I thank my mom and dad for inspiring in me the commitment to make things happen with enthusiasm. I thank them for their love, support, and encouragement for all that I do.

I thank my brothers, Tareq, Mazen, and Nader, for their undying support, and love.

I extend my sincere thanks to my cousin and sister-in-law, Sawsan, for her love, and care. She has shown care, and concern that could never be described.

I thank my cousin Johnny, for his encouragement to finish this project.

Thanks to all my friends for believing in me and showing excitement for my work. Special thanks go to Mazen, Mounir, and Armine for always being there throughout this project.

I thank Dr. F. Khendek, my supervisor, and Dr. R. Giltho, my project leader, who taught me what it takes to be a good researcher. I hope my work in general, and my thesis in particular, shows that I clearly was the one who got the training.

I thank my co-supervisor, Dr. J.W. Atwood, for his review and valuable comments.

I thank all the members in the SINTEL project at Ericsson Research. Special thanks go to Robert Huie, Brian Liu, Kindy Sylla, Andre Poulin, Jean-Charles Beaudoin, Riad Hamadi, and Clovis Muhugusa for their contributions and input in this project.

I thank my manager Denis Monette for believing in me, and for giving me the opportunity to join his team and prove myself.

TABLE OF CONTENTS

Acknowledgments vi
Table of Contents vii
List of Figures vii
List of Acronyms ix
Chapter 1: Introduction 1
1 Background 1
2 Motivations and Objectives 3
3 Thesis Contributions 4
Chapter 2: Requirements and Related work 5
1 Conferencing Scenarios 5
2 Conference Policies 9
3 Functional Requirements for Floor Control 9
4 SIP and related work from IETF 11
5 PARLAY 17
6 The Alternatives 29
7 Discussion of Existing Solutions 30
Chapter 3: Proposed Solution 32
1 Enhancement to the Parlay APIs 32
2 Floor Control Components 40
3 An Overall Architecture for Floor Control 42
Chapter 4: Use of Web Services for Floor Control 45
1 Introduction to Web Services 46
2 Exposing Floor Control as a Web Service 52
3 Enhanced Architecture with Web Services 54
4 PARLAY Mapping 67
Chapter 5: Implementation 69
1 Software Architecture 69
2 Video Conferencing Application 71
3 Graphical User Interface 75
4 The Interface 78
5 Tools and Libraries 80
Chapter 6: Conclusions 81
1 Contributions 81
2 Future Work 82
REFERENCES 83
APPENDIX A (advanced video conferencing wsdl) 88

LIST OF FIGURES

Figure 1: A screen shot of the application 6
Figure 2: Internet Protocols [19] 11
Figure 3: PARLAY business model [24] 20
Figure 4: Parlay APIs interfaces [24] 23
Figure 5: Call Control Service Packages [24] 27
Figure 6: Internet Multimedia Conferencing Architecture [31] 29
Figure 7: Centralized Conference using PARLAY 34
Figure 8: Proposed Method : revokeSpeaker() 35
Figure 9: Sequence Diagram: revokeSpeaker() 36
Figure 10: Proposed Method : floorReleased() 38
Figure 11: Sequence Diagram: floorReleased() 39
Figure 12: Flow of Events 41
Figure 13: Overall Architecture 42
Figure 14: Web Services Model 47
Figure 15: Web Service Conceptual Stack 51
Figure 16: Floor Control in a Web Service Framework 53
Figure 17: Mapping SOAP request to PARLAY API call 54
Figure 18: Use of Web Services for Floor Control 55
Figure 19: Request a floor by a participant 57
Figure 20: Grant a floor by a chairman 59
Figure 21: Grant a floor by the application logic 61
Figure 22: Revoke a floor by the application logic 63
Figure 23: Revoke a floor by a chairman 64
Figure 24: Release a floor by a participant 66
Figure 25: Software Architecture 71
Figure 26: Overall architecture of the Video Conferencing Application 72
Figure 27: A screen shot of the Conference/Floor Manager Application 75
Figure 28: A screen shot of the videoconference client application 77

LIST OF ACRONYMS

3G:	Third Generation
3GPP:	Third Generation Partnership Project
API:	Application Programming Interface
CORBA:	Common Object Request Broker
IN:	Intelligent Networks
MCU:	Multiparty Control Unit
MMUSIC:	Multiparty Multimedia Session Control
OSA:	Open Service Access
RTCP:	Real Time Control Protocol
RTP:	Real Time Protocol
SAP:	Session Announcement Protocol
SCCP:	Simple Conference Control Protocol
SDP:	Session Description Protocol
SIP:	Session Initiation Protocol
TINA-C:	Telecommunication Information Network Architecture Consortium
UA:	User Agent
UAC:	User Agent Client
UAS:	User Agent Server
UDDI:	Universal Description Discovery and Integration
UML:	Unified Modelling Language

VHE: Virtual Home Environment

WS: Web Service

WSDL: Web Service Description Language

WSFL: Web Service Flow Language

XML: Extensible Markup Language.

CHAPTER 1: INTRODUCTION

1 Background

"Advanced services, or more simply services, can be defined as anything that goes beyond two party voice calls. They are the raison d'etre of next generation networks. They are differentiating factors and crucial to service providers' survival and success. They are usually grouped under two umbrellas: session related services and non-session related services. Session related services interact with call session control while non-session related services do not interact with session control. Examples of session related services are conferencing and session forwarding and examples of non-session related services are location based services and surfing the web.

Next generation networks, 3G networks and beyond, can be conceptually divided in three separate networks: access network, core network, and service network. It is the service network that deals with the value added services related issues. It includes: The applications, which reside in application servers (A service is the end-user experience of an application) and the application enablers, which are the tools/environments/architectures, used to develop and deploy applications" [1].

Floor control enables the control of media, e.g., who can be seen and heard in a multiparty session. By multiparty session, we mean more than one party exchanging more than one media stream (e.g., audio, video, and chat) over a network. Floor control is one of the building blocks of the Internet multimedia conferencing architecture. It is required in a traditional video conferencing application as well as for multiparty session gaming.

The third generation partnership project (3GPP) [2] has selected the Session Initiation Protocol (SIP) as the sole signalling protocol for its 3G networks. SIP [3] is a proposed standard by IETF [4] published in RFC 2543. SIP initiates, modifies, and terminates multimedia sessions. These multimedia sessions include multimedia conferences, distance learning, Internet telephony and similar applications. SIP does not offer conference control services such as floor control and does not prescribe how a conference is to be managed. However, SIP can be used in conjunction with other protocols to provide floor control solutions. Multiparty Multimedia Session Control (MMUSIC) working group by IETF introduces lately an IETF draft on the "Use of SIP and SOAP[5] for Floor Control" [6] and there has been an expired draft on the "Simple Conference Control Protocol (SCCP)" [7] .

In addition to the IETF drafts, the PARLAY [8] forum aims at producing Application Programming Interfaces (APIs) that allow third party service providers to access network capabilities in a controlled, communication technology independent fashion. The telecommunications community has adopted these APIs in the context of 3GPP. The very same telecommunications community that has adopted PARLAY has also adopted SIP. This makes the exploration of venues of PARLAY and IETF related work worthwhile and relevant. Several papers have been published on PARLAY and its use in SIP Networks [9-11].

2 Motivations and Objectives

The key motivation for a floor control mechanism is controlling media streams in conferencing applications, for instance to identify which participant is allowed to send his media stream, who can be seen or heard. Floor control can avoid chaotic situations when everybody attempts to talk at the same time. In addition, floor control is needed when bandwidth restriction is a concern.

Today's mobile networks primarily use circuit switching to deliver voice and data. Tomorrow's networks, 3G networks, are built on the foundation of IP technology, which is capable of delivering multimedia services in a more efficient packet-switched manner.

The current available technologies from PARLAY and IETF do not provide a complete solution approach to implement a floor control mechanism. In this thesis, we will focus on a centralized architecture. The first step in our work is to define the requirements for a floor control mechanism, and then check if the current technologies satisfy these requirements. Based on this evaluation, we will propose an architecture that benefits from the available technologies, and which also provide the missing parts in order to achieve a complete solution.

3 Thesis Contributions

The contributions of this thesis are as follows:

1. A set of requirements for floor control. We have used them as a basis for our evaluation of the related work in PARLAY and IETF.

2. A set of proposed extensions for floor control to the current specification in PARLAY. We have submitted these extensions to the 3GPP Open Service Access (OSA) working group for standardization.

3. A novel architecture for floor control in SIP Networks integrating APIs with protocols. We have presented the work at the International Conference on Intelligence in Next Generation Networks, ICIN2003, Bordeaux, France [12].

4. A new interface for floor control as a Web Service. We have investigated the suitability of using Web Services as a step further in our study.

5. An advanced video conferencing application. We have designed and developed the application from scratch as a proof-of-concept.

CHAPTER 2: REQUIREMENTS AND RELATED WORK

Several conferencing scenarios have been studied, leading to a definition of the requirements for a generic floor control mechanism. We present briefly a description of several conferencing applications that we studied to extract the functional requirements for floor control. After defining the requirements, we surveyed the related work on SIP [3] and PARLAY [8] to identify whether they satisfy (or not) the defined requirements. Our choice of PARLAY and SIP is motivated by the key role these two technologies will play in 3G networks. SIP has been selected as the sole signalling protocol in 3GPP. In addition, PARLAY has been selected as the basis for its application development framework [24].

1 Conferencing Scenarios

In the following sections, we examine several complex scenarios for conferencing. These case scenarios are used to define the requirements for a generic floor control service that can be used for different applications.

1.1 Video Conferencing Application

Our video conferencing application has been designed and developed from scratch with the collaboration of the SITNEL project at Ericsson Research. The main goal of the application is to show the use of session and floor control capabilities extensively.

The conference is initiated using a web page as shown in Figure 1. You can add, remove participants from a conference or close the conference by clicking on the corresponding

button. You can also set your conferencing policy, for example, chair controlled or first-ask-first-served. You can allocate a time limit to hold the floor. You can assign a chairman to the conference.

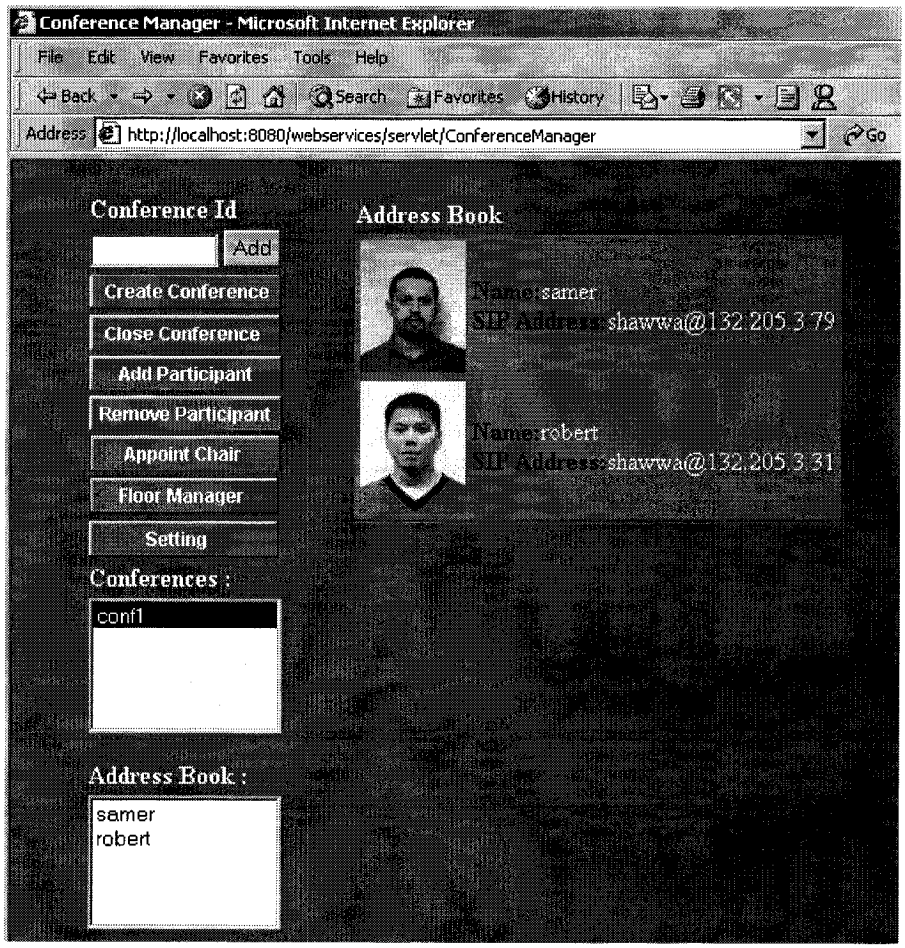

Figure 1: A screen shot of the application

Once the conference is started all participants are muted. People need first to request the floor and be granted the floor before speaking. Based on the conferencing policy the participant can

be granted (or not) the floor. An example, if the conferencing policy is chair controlled then the chairman should be able to grant (or refuse to grant) the floor explicitly. Another example, if the conferencing policy is first-ask-first-served then the floor will be granted automatically based on the queue. If the time allocated to the participant expires, the floor will be revoked automatically; he may continue talking but nobody will hear him. People can be notified when the floor state changes. For instance, which participant has the floor, which has requested the floor, and how long the floor holder can still hold the floor.

1.2 Multiparty Game Applications

"Guess What?" is the name of a multiplayer game. "Guess What?" is played over a computer network. The objective of the game is to draw pictures on a canvas that everyone in the game can see, and have members of one's team guess what is drawn within a certain time limit. Guesses are made by saying them out-loud, for all participants in the game to hear. The person drawing cannot speak, but all other players on his team may guess, possibly at the same time. Teams compete against each other in order to guess correctly more often in fewer trials.

March 2002 is an adaptation of the *Pictionary* concept for a network environment. Players may participate in the game from anywhere, using their PC connected to the Internet. The March 2002 software system must therefore provide multimedia conferencing functionality with high performance, at a low cost to the end-users.

Each user can be one of the four types: picturist, guesser, picturist-in-waiting, and listener. There can be only one picturist, only one picturist-in-waiting, but several guessers and listeners.

The picturist's objective is to get his team to guess a secret word by drawing hints on a canvas seen by all participants. The picturist must be able to hear all guesses, but not be heard if he tries to speak back to the guessers. The video stream sent to all participants in the game will show the picturist. This will allow the picturist to use facial expressions and gestures to help his team in guessing.

The Guessers are all participants of the picturist's team, excluding the picturist himself. Guessers are the only game participants that do not know the secret word; hence they try to guess it based on the picturist's hints.

The listeners must be able to hear all guesses. If they speak while the guessers are guessing, they must not be able to be heard. The ability to mute the audio output streams for each listener is given to the "picturist-in-waiting", the next picturist in the game, who is actually a member of the listening team.

1.3 TV Show Scenario

In a TV show scenario, for example a Talk Show, we can have two or more users participating in a debate or discussion. Let us say we have the host (chairman or moderator) and three participants who are allowed to speak (we may have more participants as audience). In this scenario the host is part of the conference, anybody who wants to speak and who wants his video to be received by all participants has to request the floor and the host has to grant him the floor, otherwise nobody can hear him or see him. It is possible to show on the screen of each participant the image of more than one participant at the same time. In this example, the videos are mixed and sent to all participants. At any time, a chairman may decide to revoke the

floor from a participant, e.g., he exceeded the time limit to speak. A participant may wish to release the floor if he has nothing more to add in the discussion.

1.4 Distance Learning Application

The Distance learning application enables a conference with a lecturer that presents a topic and can allow questions from the participants. The lecturer needs to know who is requesting the floor from the participants and to be able to give them the right to speak. The lecturer should be able to revoke a floor from one or more participants who holds the floor.

2 Conference Policies

Conference policies are a set of parameters and rules (e.g., maximum number of participants, need for a moderator or not, duration for speeches, etc.) that are defined at the creation of a conference. Typically, conference policies would be specified by a conference creator and need special privileges to be manipulated.

The conference policies contained in the previous section aid us to identify the floor control functionalities needed for different applications.

3 Functional Requirements for Floor Control

Based on our study for the previous conferencing scenarios we extract and define the main functional requirements for floor control.

- Request the floor: A participant should be able to request the floor.

- Grant the floor: The floor chair (or moderator) should be able to grant the floor to the requesting participant. The granted participant can have access to send his media streams.

- Release the floor: A participant should be able to release the floor and make it available to others.

- Revoke the floor: The floor chair (or moderator) should be able to revoke the floor from the participant holding the floor.

- Notification of the floor events: It should be possible to be notified about the changes in the floor status.

Our feelings are these requirements provide the complete set of functional requirements for floor control and they can fulfill any conferencing scenario that needs a floor control mechanism.

Floor control can be defined as a mechanism to control the access to the floor(s). Floor control itself does not define which media streams are floor controlled and which user has the privileges of floor chair (or moderator). Instead, some external mechanism, such as conference management is used for that. Typically, the conference owner decides which media streams are floor controlled, appoints the floor chair, and sets the conference policy. The conference owner can remove the floor anytime (so that media stream is not floor controlled anymore), or change the floor chair or floor policy.

In the next sections, we provide an overview on SIP and related work from IETF and PARLAY APIs and we discuss whether these technologies fulfill (or not) our requirements.

4 SIP and related work from IETF

SIP [3] initiates, modifies, and terminates multimedia sessions. These multimedia sessions are applicable in cases such as multimedia conferences, distance learning, Internet telephony, and similar applications. SIP does not offer conference control services such as floor control and does not prescribe how a conference is to be managed. However, SIP is used in conjunction with the other IETF protocols as shown in Figure 2 such as Session Description Protocol (SDP)[16] for session description, the Real Time Protocol (RTP) [17] for media session, the Session Announcement Protocol (SAP) [18] for session announcement. The MMUSIC has been looking at the floor control problem lately and they published the IETF draft "Use SIP and SOAP for conference floor control" [6].

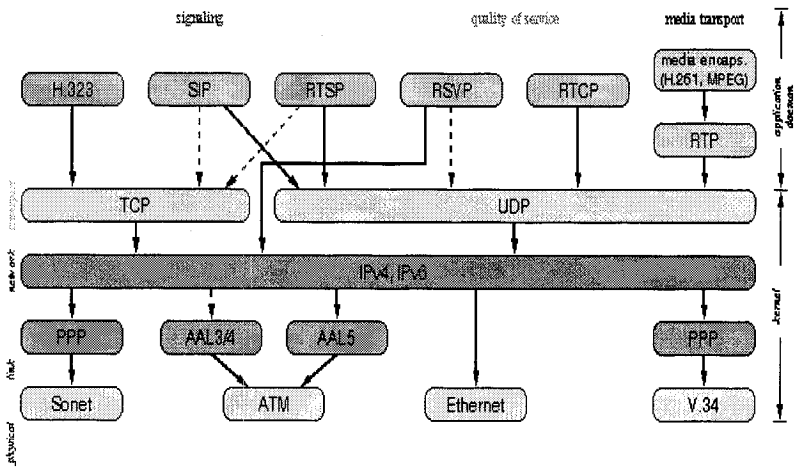

Figure 2: Internet Protocols [19]

4.1 SIP Overview

There are two components in a SIP environment: the User Agent (UA) and the network servers. The UA is the end point system that acts on behalf of a user. It consists of two parts: a client (UAC) as the caller, a server (UAS) as the called user. The UAC initiates the call and the UAS receives requests and returns responses.

SIP defines three types of network servers: the proxy, the redirect and the registrar servers. The SIP proxy behaves as UAS and UAC. It receives the incoming request from UAC and forwards it to the next proxy or UAS. Additionally, the SIP proxy has the power and the flexibility to fork incoming calls, i.e., one call can be routed to several destinations. The redirect server does not forward the request to the next server; instead it sends the redirect response to the client. The registrar server accepts registration requests. Usually it is co-located with the proxy or redirect server and may offer location services.

A SIP message consists of a request from a client to a server and a response from a server to a client. In the core SIP, the methods that can be used as request are the following: REGISTER, INVITE, ACK, CANCEL, BYE and OPTIONS. A SIP response consists of six response codes: 1XX informational, 200 success, 3XX redirection, 4XX Client error, and 5XX Server error, 6XX Global Failure. However, there are other extension methods for adding features to SIP. These extensions are additional to the core SIP as specified in the SIP-related drafts [19].

4.2 SIP Conferencing schemes

SIP supports four conference schemes: centralized, multicast, end point mixing, and full mesh [20].

- Centralized scheme

It is quite similar to the scheme used today in telecommunication networks. It involves a centralized server. This server handles both signalling and media streams. All participants are connected to it. It should be stated that the signalling part and the media part of this server could be physically separated.

- Multicast scheme

This scheme relies on a predefined multicast address and a predefined port. Participants can join the conference by connecting to the address / port. Multicast routers are used in the network. SIP is used just to inform potential participants of the existence of the conference. They can then join if they wish.

- End point mixing

The End point Mixing usually starts with a single call between two users, for example, A and B. At a later point during the call, user A invites others participants to join. New calls are created for these invitations. The Call-Ids, and tags are different from the ones used in the call with B.

User A handles both signalling and media mixing and is the only one aware of the presence of everybody. B and the new participants are aware of the presence of A only. However they can use the Real Time Control Protocol (RTCP) report from A to learn about each other.

- Full mesh

In this scheme, every participant has a signalling relationship to each and every other participant. Furthermore every participant does his own media mixing.

4.3 A More Detailed Description of the Centralized Conference

The centralized conference model is based on a central server usually called Multiparty Control Unit (MCU) or Conference Bridge. In this scheme, each participant establishes a SIP session with the central server. Between the participants, there is no signalling. The conference URL identifies the conference within the server. It is composed by the conference ID and the conference address, e.g., conference1@centralserver.com.

The media session can be managed in two ways:

- The central server can be the mixer for the session; it receives the streams from each participant, mixes them and sends them out to the other participants [21]. The conference server configuration is the same as a normal SIP UA that has the mixer and the conference capabilities added. The participant UA does not require any specific capabilities. Each participant uses the information from the RTCP [22] packet to learn about each other. The mixed stream contains the senders.

- The central server handles the signalling only. The participants have an RTP session established between them. In this case, the participant SIP UA needs to have RTP mixer capabilities added. To learn about others, each participant identifies every single stream it receives.

In the centralized conference model, the participant has two ways to join. The central server can be a dial-in server and/or a dial-out server.

- The Dial-In Conference

The Dial-in conference server allows the user to join the conference by dialling in to the server, e.g., sending a SIP INVITE message to the conference server. The conference server can accept the request or reject it for several reasons: authentication failed, conference is full. Each participant who wants to join the conference initiates a different call with a different caller ID.

- The Dial-Out Conference

In the dial-out conference, the central server invites the users to join. Generally, the participants of the conference call are known in advance. At the starting time, the conference server initiates a different call with each participant. Usually, the model supports the dial-in conference in case users already registered want to join after the conference starts up.

The advanced auto-conducted video conferencing application that we described previously [Section 1.1] is an example of application that uses a dial-out server.

4.4 SIP and SOAP for Floor Control

X. Wu et al. [6] defined an approach that uses the SIP event notification mechanism and SOAP to perform the floor control functions for centralized conferencing.

They proposed to divide floor control messages into two categories:

- A set of floor control events: These events are built upon SIP events' architecture and used to report the changes of the floor control status.

- A set of floor control commands: These commands are built using SOAP and used to change the floor control status.

They proposed to use SOAP for exchanging commands since floor control commands are in fact RPC calls and SOAP fits well for exchanging RPC calls. Nevertheless, SOAP does not provide call back methods hence the authors proposed using an event package based on the SIP Event architecture [23].

The floor control events package defines the following events:

- floor_created()

- floor_removed()

- config_changed()

- floor_changed()

- queue_changed()

The floor control commands include:

- floor_create()
- floor_remove()
- change_config ()
- floor_claim()
- floor_release()
- floor_grant()
- floor_revoke()
- remove_claims()
- reorder_claims()

5 PARLAY

The PARLAY forum aims at producing application programming interfaces (APIs) that allow third party service providers to access network capabilities in a controlled and communication technology independent manner. The telecommunications community has adopted these APIs in the context of 3GPP and they are now used as the basis for the OSA [24]. OSA is a toolkit

that can be used to realize the virtual home environment (VHE) [25], the service framework being specified by 3GPP for 3G networks.

The PARLAY forum was created in 1998 as a closed forum by a handful of players from the telecommunication and the information technology industries. It was extended to a few more members in 1999, then became an open forum in 2000. Most key players of these two industries are now members. The forum works closely with other industry groups such as 3GPP and the principal objective is the opening up of telecommunication networks in a secure manner.

PARLAY APIs provide secure access to (and secure control of) a selected range of network capabilities for application development purposes. They are open, signalling protocol neutral, and object oriented. Unified Modelling Language (UML) [26], a technology independent modelling language, is used for their specification. They can be deployed using either the Object Management Group (OMG) Common Object Request Broker (CORBA) [27] or Microsoft DCOM [28].

The business model behind them aims at allowing new players in the arena of service provisioning including third parties in the realm of portable application development. The section starts by introducing the model. After that we provide a general overview of the APIs, followed by a more detailed description of the call control APIs.

The call control service exposes call control capabilities, which include floor control. We describe it in more detail in the last section.

5.1 Introduction to the business model

The PARLAY business model is deeply rooted in the Telecommunication Information Network Architecture Consortium (TINA-C) business model. In TINA-C, there are five business roles [29]: consumer, retailer, third party provider, connectivity provider and broker. These roles are conceptual and the same entity can play several roles at the same time.

The consumer role is further divided in two roles: end-user and subscriber. In business settings, employers usually play the role of subscribers by subscribing to telecommunications services (e.g., conferencing, fax) while employees play the role of end-users by consuming the services. In private settings, the same entity plays both roles in most cases.

The retailer is the one stop shop where the subscriber subscribes to all services. The retailer does not necessarily produce the services it offers to the subscribers. It can get them from third party providers. The connectivity provider owns the network in which the services are realized. The broker ensures that relevant information is available to all actors and potential actors.

In the PARLAY business model, the word "service" has a different meaning. Services are no longer value-added services consumed by end users. They are network capabilities exposed for application development purposes. As in TINA-C, the same PARLAY entity can play several roles at the same time. These roles are briefly introduced below.

- The client application: It is the consumer of PARLAY services and is the equivalent of the end user in TINA-C.

- The enterprise operator: It is the entity that subscribes to PARLAY services and is the equivalent of the subscriber in TINA-C. The services to which an enterprise has subscribed can be used by several client applications. Both the client application and the enterprise operator are consumers.

- The framework operator: The enterprise operator needs to subscribe to the PARLAY services, prior to the usage by its client applications. The framework operator is the equivalent of the retailer in TINA-C. It makes the PARLAY services available for subscription.

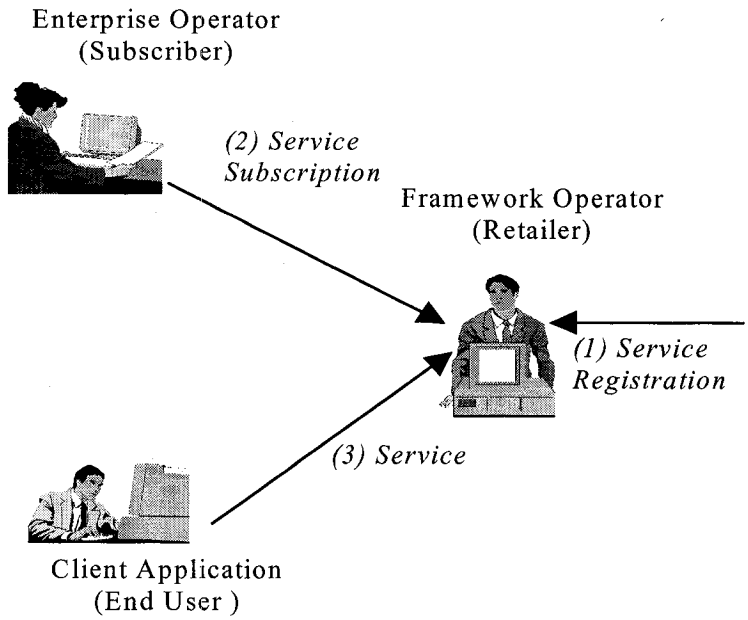

Figure 3: PARLAY business model [24]

Figure 3 illustrates how the business model works in practice:

- First step - Service Registration: The PARLAY services register to the framework. This step is needed because the services may not be in the framework domain. They may belong to third parties.

- Second step - Service Subscription: This step is needed in order to allow the client applications to access the services. It involves the enterprise operator who acts as subscriber and the framework operator who acts as retailer.

- Third step - Service Usage: This step is needed in order to allow the client applications to use the services. It includes authentication and involves the framework and the client applications.

The connectivity provider and broker do not appear explicitly in the PARLAY model. However, they are implicitly defined. The connectivity provider owns the network capabilities that are exposed. As in TINA-C, the broker makes relevant information available to all actors and potential actors.

Beyond opening up telephony networks in a secure manner to third parties for application development purposes, the model allows new players in the telecommunications world:

- An entity that does not own any network can establish itself as an enterprise operator; develop applications that it can make available as advanced services to subscribers.

- An entity that does not own any network and that does not develop applications can establish itself as a framework operator to act as a one stop shop towards enterprise operators.

5.2 An overview of the APIs

The Parlay APIs offer a gateway/server side and an application/client side. The gateway/server exposes the network capabilities and implements the interfaces the application/client expects to find. The application/client side implements the call back interfaces that allow the gateway/server to inform the application/client about what happens in the network.

Synchronous and asynchronous modes of communications are supported. The APIs are grouped under two umbrellas: framework interfaces, and service interfaces. The framework interfaces are the entry door to the network capabilities while the service interfaces expose the actual capabilities. Framework and service interfaces are successively introduced below. Figure 4 provides a graphical illustration.

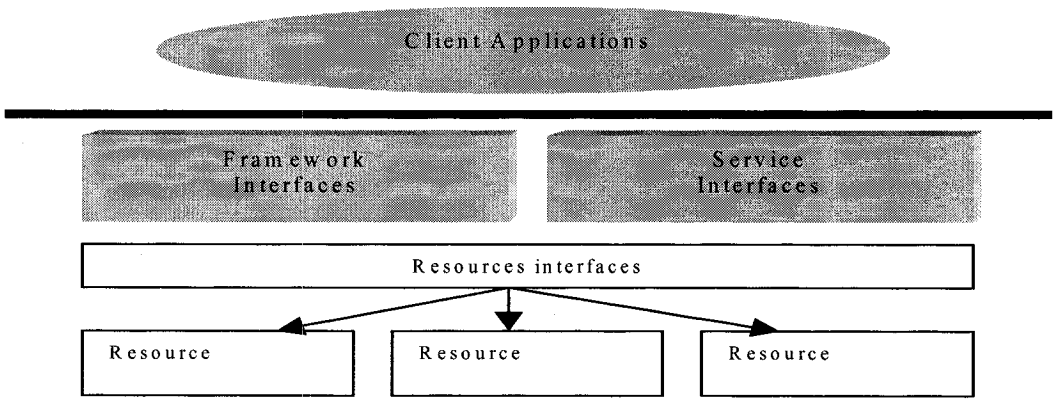

Figure 4: Parlay APIs interfaces [24]

5.3 The framework interfaces

The framework interfaces provide the capabilities necessary to make the service interfaces open, secure, and resilient. They consist of:

- Trust and Security management: It is the contact point through which a client application gains access to the framework. It provides authentication and authorization. The authentication model is peer to peer. The client application must authenticate itself prior to the usage of services. However, it is up to the client application to decide whether the framework should authenticate itself.

- Event Notification: It provides a notification mechanism allowing the framework's events to be sent to applications. Notifications are sent for instance when services become available or unavailable.

- Service Discovery: It provides information on the services that are available. It also provides search capabilities.

- Service Registration: It allows new services to register with the framework. The framework will then notify the interested client applications.

- Service Subscription: It allows enterprise operators to subscribe to the registered services.

- Integrity Management: It monitors and controls the events and operations that may affect the network integrity. It offers features such as load management, fault management, operation and maintenance.

5.4 The service interfaces

The service interfaces defined so far are the following:

- Call Control: It exposes the capabilities for call creation, manipulation and tear down. It is the most relevant in SIP networks as already stated.

- User Interaction: It allows applications to interact with end-users. Applications may play files to specific users or record files from specific user(s). They can also perform text-to-speech conversion.

- Generic Messaging: It is used by client applications to receive, send and store messages. The message can be electronic mail or voice mail.

- Mobility: It provides a general geographic location service. It offers capabilities to allow client applications to obtain the geographical location and the status of fixed, mobile and IP based telephony users

- Terminal Capabilities: It enables applications to retrieve the capabilities of given terminal(s). The terminal can be a mobile phone, a personal digital assistant (PDA), or a computer connected to a network.

- Connectivity Management: It provides tools for the enterprise operator or client applications to set up a Provisioned Quality of Service (QOS) in the operator network. Client applications such as video conferencing can select the optimized QOS by specifying the delay, loss, jitter and transfer rate between two endpoints in the network.

- Account Management: It allows applications to enable or disable charging-related event notifications and to query account balances.

- Charging Service: It is used by applications to charge the end user for the usage of the applications. For example, an application can charge a user for a streamed video.

- Data Session Control: It provides a means to control data sessions between terminals attached to a network. When a terminal requests the establishment of a data session the application can reject the request, pursue the establishment as requested by the terminal or pursue it but with a destination different from the one requested by the terminal.

- Presence and Availability Management: It allows the viewing and the management of presence and availability information. It also allows the subscription to presence and availability related events.

5.5 A More Detailed Description of The Call Control Service

The call control service exposes call control capabilities as already stated. A client application can create and route a call through the network. It can also request notifications from a call initiated in the network. The notifications are based on call-related events such as ANSWER, and BUSY. We successively present the call model on which the call control service is based, and the packages that make it.

- The call model

The Parlay call model is reminiscent of Intelligent Networks (IN) and circuit switched telephony. It is very close to the model used by the Java Telephony API [30]. This model is composed of:

Call: It is an abstraction of the physical call in the network. It is used to establish the relation between parties involved in a call. Several call objects can represent the same physical call, each one of them exposing a different view.

Call Leg: It represents a logical association between the call and an address. The association is made when the call is routed to a specific address. This object is similar to the connection object of JTAPI.

Address: It represents a party in a call.

- The packages

The call control service defines four packages as shown by Figure 5.

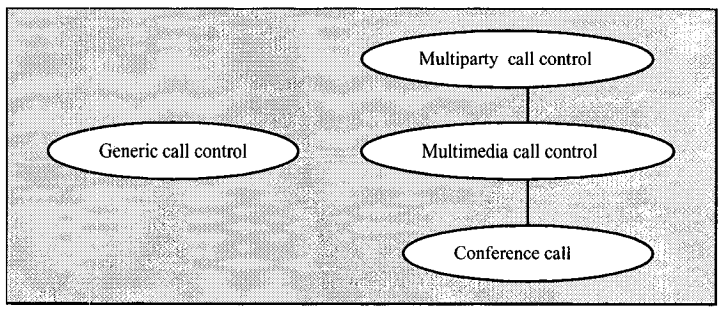

Figure 5: Call Control Service Packages [24]

There is a generic call control package, and three other packages that are tiered from multiparty to conferencing. The packages are succinctly introduced below.

The Generic Call Control Service: It provides basic call control capabilities and supports only two-party calls. It does not allow the manipulation of call legs and none of the three other packages builds on it.

The Multiparty Call Control Service: It provides multiparty call related functionality. Client applications have access to call leg operations and can establish calls, where any given number of legs can be simultaneously connected.

The Multimedia Call Control Service: It enhances the multiparty call control service with multimedia functionality. The media stream concept is introduced. A media stream is a bi-

directional media connection associated with a leg and negotiated with the terminal used by the end user. Basically, the media stream represents the different media types supported (e.g., audio, video).

The Conference Call Control Service: It enhances the multimedia call control service by adding conferencing functionality.

5.6 Use of PARLAY API for Floor Control

The PARALY APIs provide floor control functionalities as part of the Conference Call Control Service, which builds upon the multiparty call control service. The key goal of the Conference Call Control Service is to enable applications to create centralized conferences and to manipulate sub-conferences. The generic conference also gives the possibility for media manipulation and resource reservation in addition to floor control.

Our goal is to determine how well PARLAY APIs can be used to provide access to floor control functionalities. We will identify the PARLAY APIs methods for floor control that have been specified and evaluate these methods with respect to our requirements.

The specified PARLAY methods that we have identified for floor control implementations are:

- inspectVideo()
- inspectVideoCancel()
- appointSpeaker()

- selectChair()
- changeConferencePolicy()
- chairSelection()
- floorRequest()

6 The Alternatives

IETF started the standardization of conference protocols in 1996. The Internet Multimedia Conferencing Architecture [31] was proposed as a framework for the design of conferencing applications for the Internet. Most of the proposed protocols of this framework as shown in Figure 6 have been standardized and some are still in the standardization process.

This proposal reflects the work that has been done and is currently ongoing in the IETF working group MMUSIC as standardization for conference management and setup.

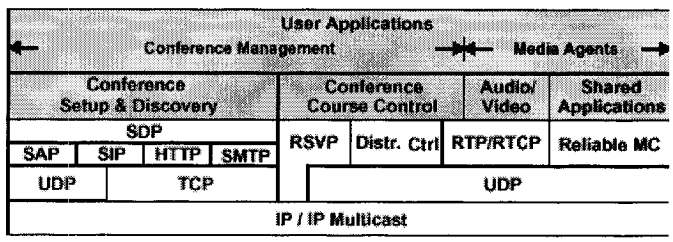

Figure 6: Internet Multimedia Conferencing Architecture [31]

Borman et al. proposed the Simple Conference Control Protocol for tightly coupled conferences. They propose the same service as proposed for an H.323 [32] system, however, with higher scalability.

SCCP provides the following floor control services:

- grabFloor()
- inhibitFloor()
- releaseFloor()
- testFloor()
- askFloor()
- giveFloor()
- holderOfFloor()

7 Discussion of Existing Solutions

The proposed drafts from IETF met our defined requirements. However, one limitation of the IETF drafts is the lack of an API that provides access to the protocol functionalities.

APIs provide major advantages such as a mechanism that allows isolating developers (designers and programmers) working at different layers, which allows them to work independently. In addition, APIs usually provide backward compatibility to older low-level

implementations and well as enabling a faster development path. With the lack of APIs, people working at different levels will be dependent on each other, backward compatibility will not be respected, and the development time will increase.

The specified methods in Parlay APIs for floor control have met three of our requirements (request the floor, grant the floor, and notification of floor changes). The Parlay Conferencing API did not specify any methods for releasing or revoking the floor.

A proposed enhancement to the current specification of the PARLAY API is needed by proposing extensions for floor control. An overall architecture combining the SIP protocol with the PARLAY API is needed for a complete solution.

CHAPTER 3: PROPOSED SOLUTION

Through the literature review, we have learned that the SIP protocol can support several conferencing schemes including the centralized conference. Furthermore, PARLAY call control service is based on the centralized conference model. Both technologies provide floor control functionalities. However, we have identified that PARLAY as per today's specification does not fulfill all the requirements for floor control defined in the previous chapter. In addition, we have identified the need for a complete solution for floor control integrating APIs with the protocols.

In this chapter, we introduce the proposed enhancement to the PARLAY APIs. We identify the main components in a Floor Control mechanism and their role in a multiparty session. We describe how these components interact with each other. We also propose an overall architecture for floor control.

1 Enhancement to the Parlay APIs

Based on our evaluation of the current specification in 3GPP OSA/PARLAY APIs and the defined requirements for floor control, we identified extensions for floor control to the current specification. The proposed extensions have been submitted for standardization to the 3GPP OSA working group.

The proposed methods are:

- revokeSpeaker() : This method revokes the floor from the participant in the conference who has the floor. The participant is then muted.

- floorReleased() : This method is used to inform the application about the floor release from the network.

As we mentioned before in the literature review, the Parlay APIs offer a gateway (server side) and an application (client side). The gateway exposes the network capabilities and implements the interfaces the application expects to find. The application implements the call back interfaces that allow the gateway to inform the application about what happens in the network.

Generally, in a conferencing system, there are three components, the conference application, the conference bridge and the terminals as shown in Figure 7. The conference bridge exposes the network capabilities and implements the conference call control service API. The conference application calls the methods that are supported by the conference call control service and implements the call back interfaces that allow the conference server to inform the application about what happens in the network.

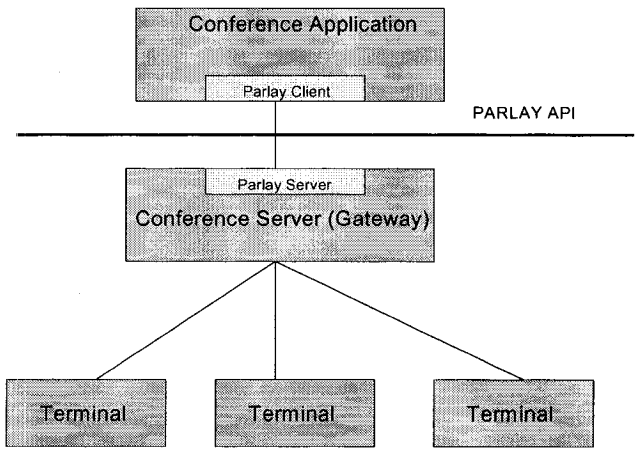

Figure 7: Centralized Conference using PARLAY

1.1 Proposed method in PARLAY API: revokeSpeaker()

Based on the floor control requirements defined in the previous chapter it should be possible to revoke a floor from a participant. This requirement proposes a new method as shown in Figure 8 to be added to the SubConference Call Interface to allow applications to have the ability to revoke the floor from the participant in the conference who has the floor. Whether this method can be used depends on the selected conference policy. For instance, if the application allocates a maximum speech time to participants, then when the time elapses the application should be able to revoke the floor from the participant in the conference who has the floor.

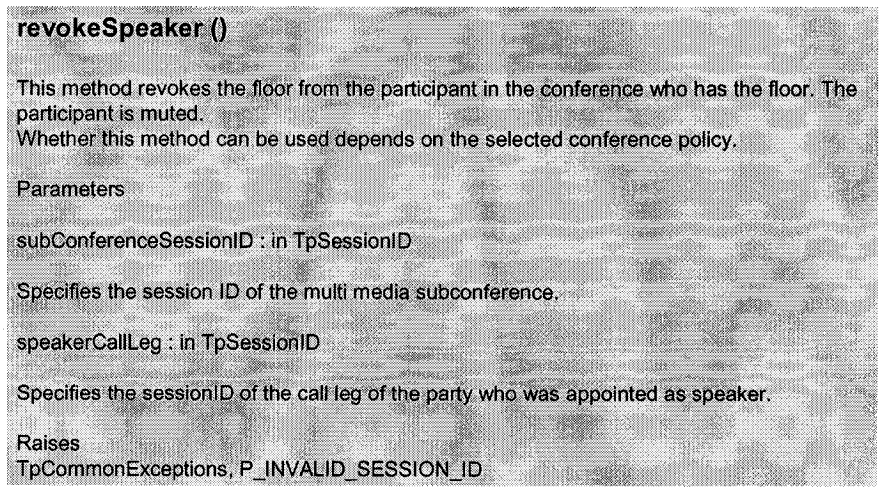

Figure 8: Proposed Method : revokeSpeaker()

The sequence diagram in Figure 9 illustrates the use of the revokeSpeaker() method in a conference application. We show how an application can revoke a floor from a speaker when the maximum speech time elapses. Once a floor is revoked from the speaker he cannot talk anymore.

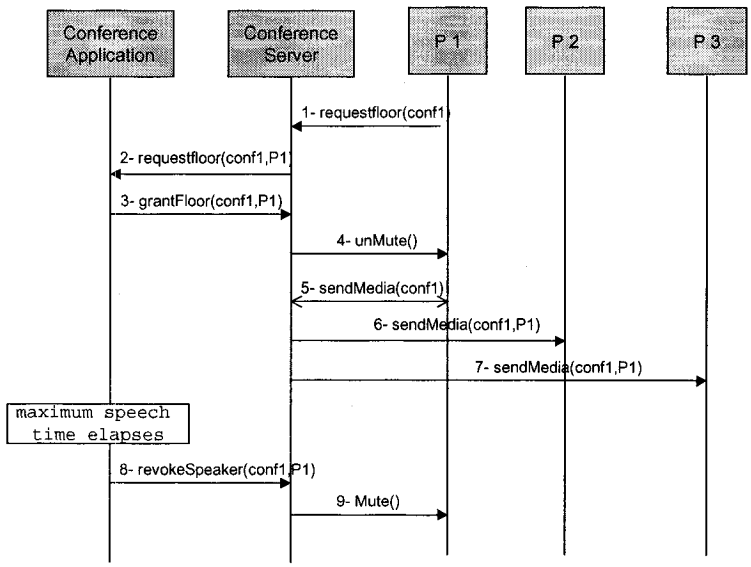

Figure 9: Sequence Diagram: revokeSpeaker()

Pre-condition: The conference server has already established a conference among the parties P1, P2, and P3. All participants are muted and they are not allowed to speak unless they request the floor and the floor is granted to them

1: P1 requests the floor from the conference server.

2: The conference Server forwards the request to the application logic.

3: The application logic decides to grant the floor to P1. It sends the request to the conference server.

4: The conference server un-mutes P1.

5: P1 sends his media stream to the conference server.

6-7: The conference server sends the media stream to P2, P3

8: The application logic decides to revoke the floor from P1 since the maximum time for the speech elapses. It sends the request to the conference server.

9: The conference server mutes P1. P1 is not allowed to send his media stream anymore.

1.2 Proposed method in PARLAY API: floorReleased()

Based on the floor control requirements defined in the previous chapter it should be possible for a participant to release a floor and to send notification about the changes in the floor status. These requirements propose a new method as shown in Figure 10 to be added to the AppConference Call Interface to notify the application about the floor releases from the network. Based on the conferencing policy the application can take the next action. For instance, if the conferencing policy is first-ask-first-served the application logic could grant the floor to the next one in the queue after being notified about the release of the floor from the floor holder.

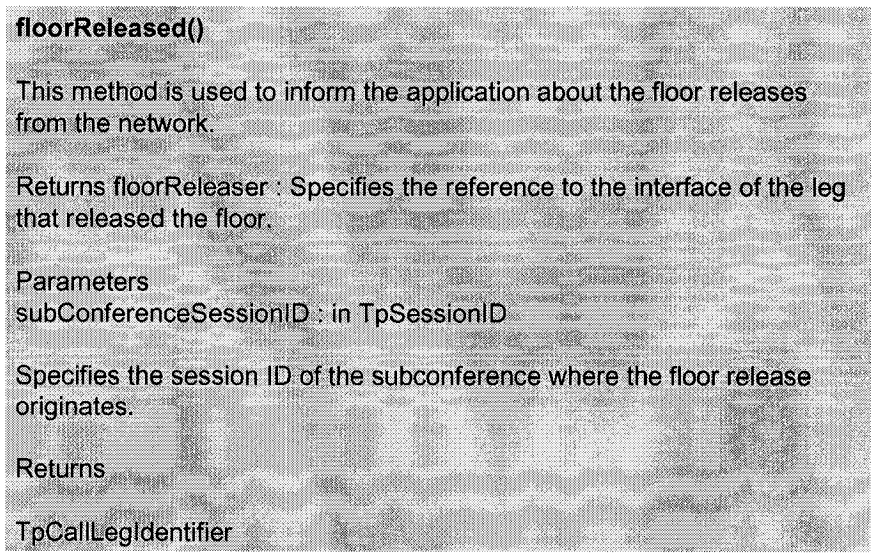

Figure 10: Proposed Method : floorReleased()

The sequence diagram in Figure 11 illustrates the use of the floorReleased() method in a conference application. We show how the conference server notifies the application about the releases of the floor from the network.

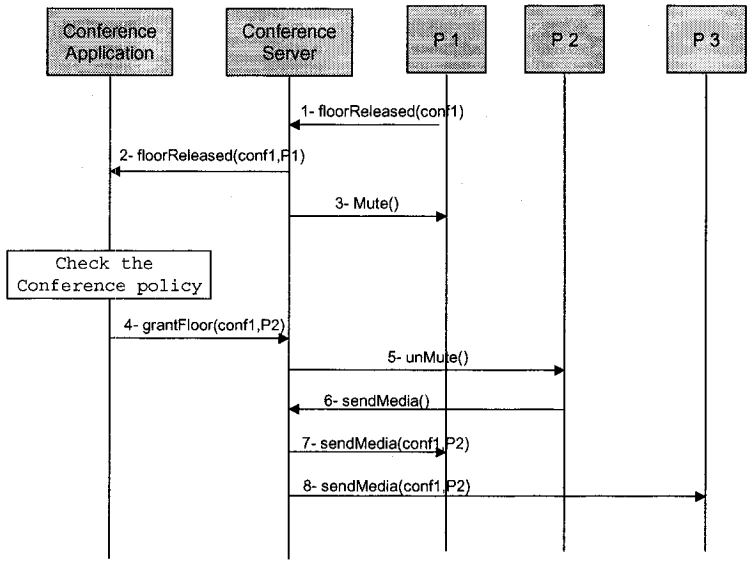

Figure 11: Sequence Diagram: floorReleased()

Pre-condition: The conference server has already established a conference among the parties P1, P2, and P3. P1 has the floor

1: P1 wishes to release the floor and he sends the request to the conference server.

2: The conference Server notifies the application about the release from the network.

3: The conference Server mutes P1.

4: The application logic decides to grant the floor to P2 based on the logic. It sends the request to the conference server.

5: The conference server un-mutes P2.

6: P2 sends his media stream to the conference server.

7-8: The conference server sends the media stream to P1, P3

2 Floor Control Components

We decomposed the functionality of the floor control mechanism into three main components:

Floor Requester: The Floor Requester requests services provided by the floor control mechanism, e.g., request a floor, and/or release a floor.

Floor Manager: The Floor Manager makes decisions about the floor requests according to the policy and decides which streams can be seen or heard.

Floor Processor: The Floor Processor communicates with the Floor Requester/Manager to realize the floor requests/responses and process the actions required based on the decisions taken by the Floor Manager, e.g., which streams are to be mixed together.

In Figure 12, we show the interactions among the main components of the floor control. The conference setup is assumed to already exist among the three participants (P1, P2, and P3). The floor policy for the scenario below is based on first-ask-first-served.

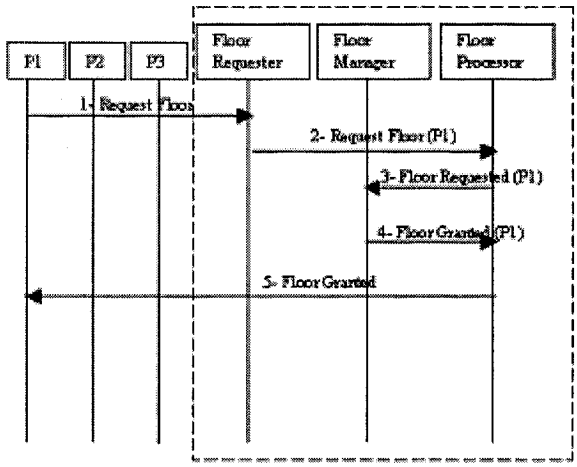

Figure 12: Flow of Events

Description of Events:

1. Participant 1 asks the Floor Requester to send a floor request.

2. The Floor Requester sends a floor request to the Floor Processor.

3.The Floor Processor informs the Floor Manager about the request.

4. The Floor Manager grants the floor to the Participant and informs the Floor Processor.

5. The Floor Processor informs the participant the floor has been granted and performs the necessary actions for media processing.

In the next section, we show the overall architecture for the floor control. By dividing the floor control into three components, we provide flexibility in locating the different components in different entities.

3 An Overall Architecture for Floor Control

The overall architecture is given in Figure 13. The MCU acts as a bridge between the Application Server and the SIP clients. The Media Part is the entity that receives, mixes, and distributes all media streams in a conference. The resources in the media part are controlled by the Control Part entity. The Control Part performs the MCU signaling functionality. It is responsible for conference call creation, routing, and management. It also gives the applications the ability to manipulate conferences through the PARLAY APIs.

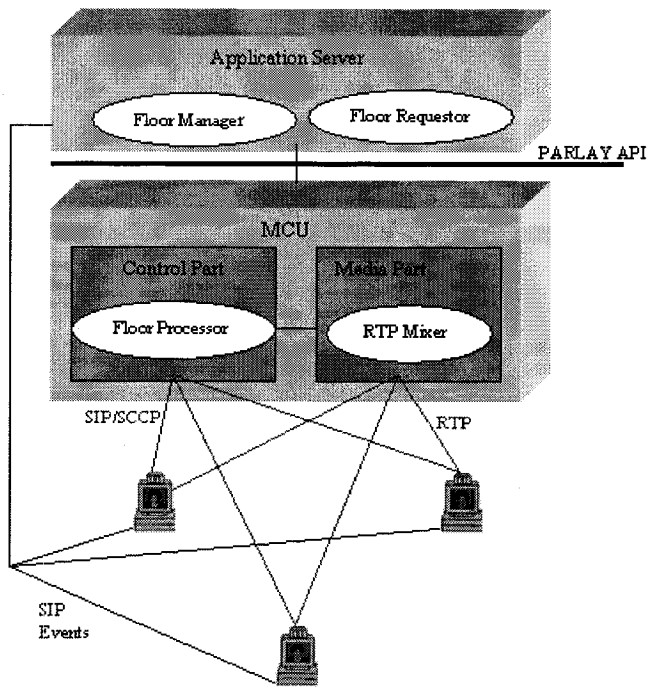

Figure 13: Overall Architecture

There exist different possible locations for the floor control components with respect to the centralized conferencing model. Since the resources of the media are controlled in the Control Part, we propose to locate the Floor Processor in the Control Part of the MCU, as this entity is

responsible for processing the media changes. This extends to be a logical placement especially with the MCU acting as a bridge between the Application Server and the clients.

As for the Floor Manager component, there are two options:

1. Co-locate the Floor Manager with the Floor Processor in the MCU, or

2. Locate the Floor Manager in the Application Server.

With the first option, the messages between the MCU and application server are reduced. However, the second choice gives the flexibility to implement different floor control scenarios if we consider that the MCU and the application server are two separate entities and these entities can be located in different domains.

Lastly, the third component, Floor Requester, can be located either in the Application Server or on the Client. Adding Floor Requester to the Client site adds the load on the end user. However, if we add the Floor Requester to the Application Server, the server can provide the Floor Requester as a remote application and hence reduce the load on the client site.

As for interfaces, the interface between the Application Server and MCU is PARLAY/CORBA. The interface between the Control Part and Media Part is the MEGACO based API. We use RTP to carry real-time media over IP between the Media Part and the clients. We use SIP for signaling and the SIP Events to report changes about the floor status.

In conclusion, we can say that the SIP protocol can be used along with the PARLAY API and other protocols to offer a complete solution for floor control. We have shown this in the

proposed architecture along with the proposed extensions for floor control in the PARLAY API.

CHAPTER 4: USE OF WEB SERVICES FOR FLOOR CONTROL

In the previous chapter, we proposed extensions to the PARLAY APIs. We identified the main components in a Floor Control mechanism and their role in a multiparty session. We described how these components interact with each other. We also proposed an overall architecture for floor control combining protocols with APIs.

We have learned so far that PARLAY as per today's specification is not easy to grasp by developers without circuit switched telephony and IN background. The same developers may also find the level of abstraction to be too low. These two issues motivate us to explore emerging technologies (i.e., Web Services) as a step further in our study. We have investigated the suitability of using Web Services to offer floor control functionalities with a higher level of abstraction than the PARLAY API.

In this chapter, we start by an overview on Web Services framework and present our solution for a Floor Control solution in a Web Service framework. We underline that this solution complements the solution described in the previous chapter. We used the proposed architecture in the previous chapter as a basis for the Web Service solution to provide a higher level of abstraction than the PARLAY API, and enable easy integration with other applications.

1 Introduction to Web Services

Web Services are becoming the de facto framework for application-to-application interactions on the Web. A Web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging. A web service is described using a standard, formal XML notation, called its service description. It covers all the details necessary to interact with the service, including message formats, transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfill a specific task or a set of tasks. They can be used alone or with other web services to carry out a complex aggregation or a business transaction.

1.1 Web Services Model

The Web Services architecture is based up the interactions between three roles: service provider, service registry and service requestor. The interactions involve the publish, find and bind operations. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web Service). The service provider defines a service description of the Web Service and publishes it to a service requestor or service registry. The service requestor uses the find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web Service implementation. Service provider and service requestor roles are

logical concepts and a service can exhibit characteristics of both. In Figure 14 we illustrate these operations, the components providing them and their interactions.

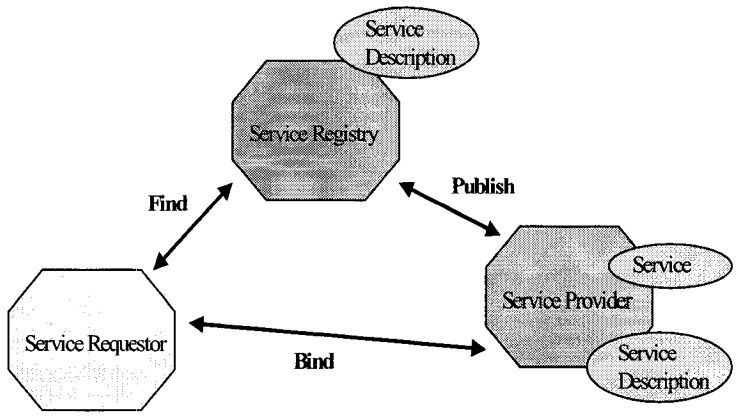

Figure 14: Web Services Model

1.2 Roles in a Web Service Architecture

- Service provider: From a business perspective, this is the owner of the service. From an architecture perspective, this is the platform that hosts use to access the service.

- Service requestor: From a business perspective, this is the business that requires certain functions to be satisfied. From an architecture perspective, this is the application that is looking for and invoking or initiating an interaction with a service. The service requestor role can be played by a browser driven by a person or a program without a user interface, for example another Web Service.

- Service registry : This is a searchable registry of service descriptions where service providers publish their service descriptions. Service requestors find services and obtain

binding information (in the service descriptions) for services during development for static binding or during execution for dynamic binding.

1.3 Operations in a Web Service Architecture

For an application to take advantage of Web Services, three behaviors must take place: publication of service descriptions, lookup or finding of service descriptions, and binding or invoking of services based on service description.

Publish: For a service to be accessible, a service description needs to be published so that the requestor can find it.

Find: In the find operation, the service requestor retrieves a service description directly or queries the service registry for the type of service required.

Bind: In the bind operation the service requestor invokes or initiates an interaction with the service at runtime using the binding details on the service description to locate, contact and invoke the service

1.4 Artifacts of a Web Service

- Service: The implementation of a Web Service is the service. A service is a software module deployed on network-accessible platforms provided by the service provider. It exists to be invoked by or to be interact with a service requestor. It also can function as a requestor, using other Web Services in its implementation.

- Service Description: The service description contains the details of the interface and implementation of a service. This includes its data types, operations, binding information and network location. It could also include categorization and other metadata to facilitate discovery and utilization by service requestors. The service description might be published to a service requestor or to a service registry.

1.5 Web Service Development Lifecycle

The Web Services development lifecycle includes the design, deployment, and runtime requirements for each of the roles: service registry, service provider and service requestor. Each role has specific requirements for each element of the development lifecycle.

The development lifecycle can have four phases:

- Build: The build phase of the lifecycle includes development and testing of the Web Service implementation, the definition of the service interface description and the definition of the service implementation description. Web Service implementations can be provided by creating new Web Services, transforming existing applications into Web Services, and composing new Web Services from other Web Services and applications.

- Deploy: The deploy phase includes the publication of the service interface and service, implementation definition to a service requestor or service registry and deployment of the executables for the Web Service into an execution environment, typically, a Web application Server.

- Run: During the run phase, the Web Service is available for invocation. At this point, the Web Service is fully deployed, operational and network-accessible from the service provider. Now the service requester can perform the find and bind operations.

- Manage: The manage phase covers ongoing management and administration of the Web Service application, such as, security, performance, quality of service and business processes.

1.6 Web Service Conceptual Stack

Figure 15 depicts a conceptual Web Services Stack necessary to perform the publish, find and bind operations in an interoperable way. The upper layers build upon the capabilities provided by the lower layers. The vertical towers represent standard requirements that must be addressed at every level of the stack. The text on the left represents standard technologies that apply at that layer of the stack.

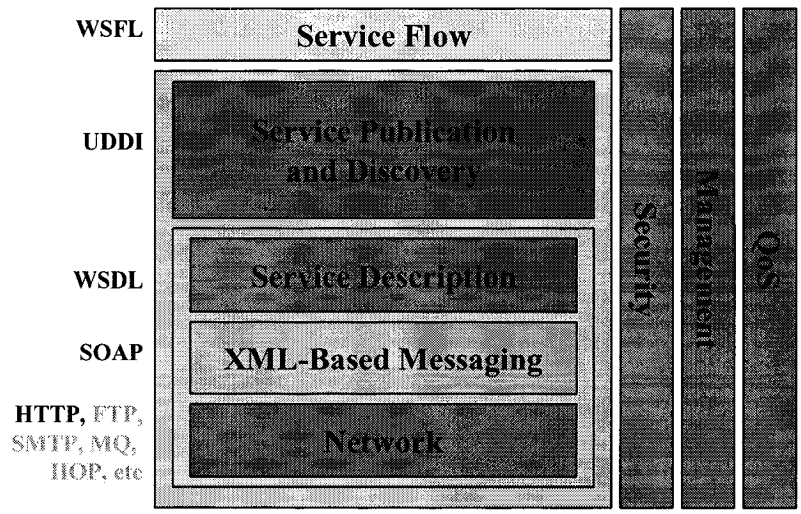

Figure 15: Web Service Conceptual Stack

The foundation of the Web Services stack is the network. Web Services must be network-accessible to be invoked by a service requestor. Web Services that are publicly available on the Internet use commonly deployed network protocols. HTTP is the de facto network protocol for Internet-available Web Services.

The next layer, XML-based messaging, represents the use of XML as the basis for the messaging protocol. SOAP is the chosen XML messaging protocol for many reasons:

- It is a standardized enveloping mechanism for communicating document-centric messages and remote procedure calls using XML.

- It is basically an HTTP POST with an XML envelope as payload.

- It is preferred over simple HTTP POST of XML because it defines a standard mechanism to incorporate orthogonal extensions to the message using SOAP headers and a standard encoding of operations or function.

- SOAP messages support the publish, find and bind operations in the Web Services architecture.

The service description layer is actually a stack of description documents. First, WSDL is the de facto standard of the XML-based service description. This is the minimum standard service description necessary to support interoperable Web Services. WSDL defines the interface and mechanism of service interaction. The WSDL document can be complemented by other service description documents to describe these higher level aspects of the Web Service. For example, business context is described using UDDI data structures in addition to the WSDL document. Service composition and flow are described in a Web Services Flow Language (WSFL) document.

2 Exposing Floor Control as a Web Service

In Figure 16, we show how a service provider hosting an implementation of the floor control Web Service. The service provider defines the WSDL description of the Web Service and publishes it in a service registry. The service requestor uses the find operation to retrieve the floor control WSDL file from the service registry and uses it to bind with the service provider and invoke or interact with the floor control implementation.

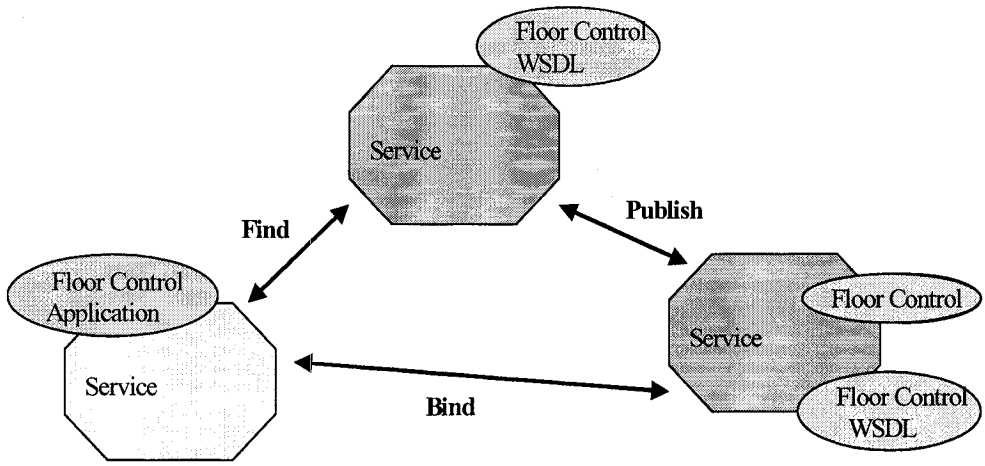

Figure 16: Floor Control in a Web Service Framework

In Figure 17, we show a service requestor implements a floor control application using the floor control WSDL file to bind with the service provider and invoke the methods supported by the floor control Web Service. In this case the floor control invokes the grantFloor() SOAP message which translates internally to a PARLAY API call to connect to the MCU over CORBA. The MCU sends the corresponding SIP messages to the clients.

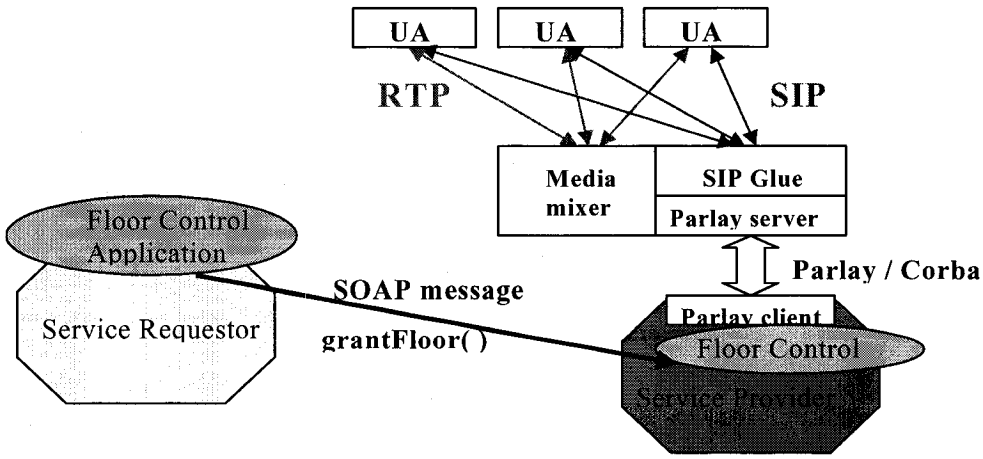

Figure 17: Mapping SOAP request to PARLAY API call

3 Enhanced Architecture with Web Services

In Figure 18, we show an advanced conference/floor manager application in the Internet domain, which uses the advanced video conferencing application web service published by the operator in the UDDI registry. The advanced conference/floor manager application is capable of calling the methods that are specified in the WSDL of the Web Service. The advanced video conferencing application web service implements floor control capabilities in a conferencing scenario. The floor control capabilities are briefly discussed below:

- requestFloor(): This method is used by a participant to request the floor.

- grantFloor(): This method is used to grant the floor to the participant requesting the floor. Whether this method is used by a participant (acting as a chairman) or not depends on the conferencing policy. Once the floor is granted the participant can have access to send his media streams.

- releaseFloor(): This method is used by a participant to release the floor and make it available to others. Once the floor is released the participant has no access to send his media streams anymore.

- revokeFloor(): This method is used to force a release from the floor holder. Whether this method is used by a participant (acting as a chairman) or not depends on the conferencing policy. Once the floor is revoked the participant who was holding the floor has no access to send his media streams anymore, even if he talks nobody can hear him.

- subscribeEvents(): This method is used to subscribe to the floor control events and be notified of the changes in the floor. For example, the changes in the list of the floor holders and the floor requests.

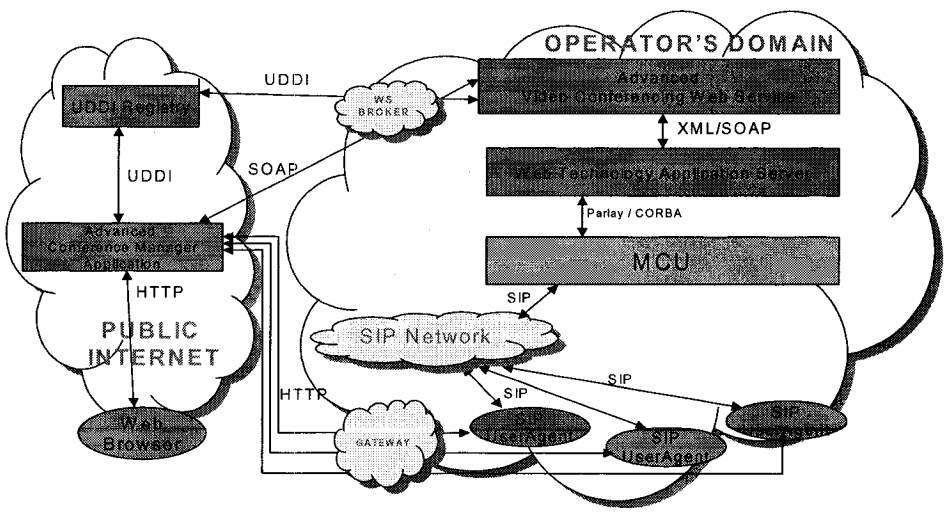

Figure 18: Use of Web Services for Floor Control

Each method described in the advanced video conferencing application web service is specified by a line or so of XML and transported over SOAP to the Web technology application server. The Web technology server analyses it then maps onto the appropriate PARLAY API calls and sends it to the MCU using CORBA. MCU executes the call and sends the appropriate requests to Video Conference Clients over SIP. The Video Conference Clients are capable of sending floor control commands over HTTP towards the Conference Manager application and also are notified about the changes in the floor status.

Scenario 1: Request a Floor by a participant

In Figure 19, we show the interaction between the main components. The conference setup is assumed to already exist among the three participants (P1, P2, and P3).

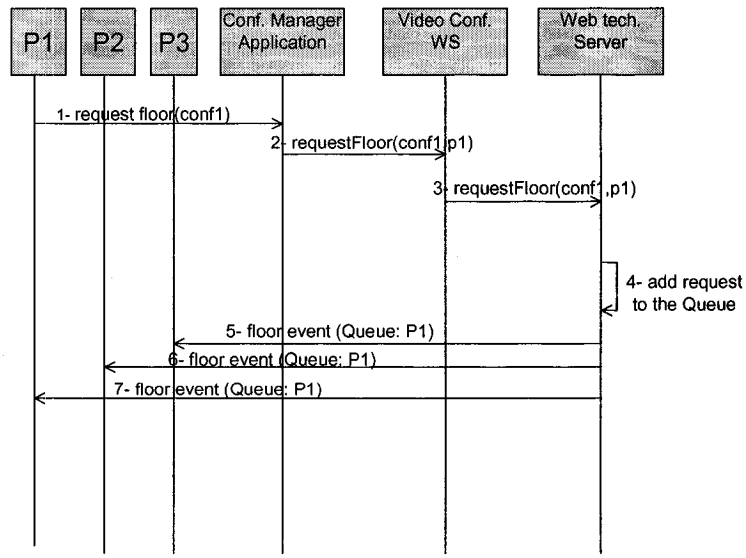

Figure 19: Request a floor by a participant

Description of Events:

1. Participant 1 sends an HTTP POST to request a floor from the Conference/Floor Manager Application

2. The Conference/Floor Manager Application maps the request to XML format and sends the request to the Video Conferencing Application Web Service

3. The Video Conferencing Application WS send the data using SOAP to the Web Technology Application Server

4. The Web Technology Application Server (implements the advanced video conferencing application WS) analyses the message and adds the request to the queue.

5-7. The Web Technology Application Server notifies all participants with an update about the list of the floor requests.

Scenario 2: Grant a Floor by a chairman

In Figure 20, we show the sequence diagram for granting a floor to a participant by a chairman. The conference setup is assumed to already exist among the three participants (P1, P2, and P3), participant 1 has requested the floor, participant 2 has the role of a chairman (or floor manager) and has the right to grant (or not) the floor to all parties requesting the floor. The floor control policy is assumed to be chair-controlled.

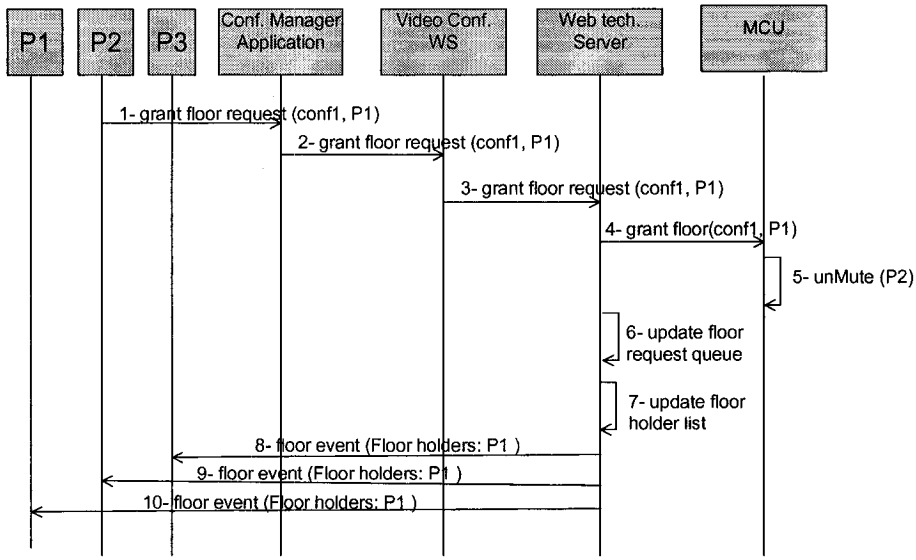

Figure 20: Grant a floor by a chairman

Description of Events:

1. Participant 1 sends an HTTP POST to request a floor from the Conference/Floor Manager Application.

2. The Conference/Floor Manager Application maps the request to XML format and sends the request to the video conferencing application web service.

3. The Video Conferencing Application WS sends the data using SOAP to the Web Technology Application Server.

4. The Web Technology Application Server (implements the advanced video conferencing application WS) analyses the message and decides to grant the floor to participant 1 based on the chairman's request, maps it to a PARLAY API call, and sends it to the MCU using CORBA.

5. The MCU, internally, asks the media manager to un-mute participant 2.

6-7. The Web Technology Application Server updates the floor requests and floor holders' lists.

8-10. The Web Technology Application Server notifies all participants with an update to the floor requests and floor holders' lists.

Scenario 3: Grant a Floor by the application logic

In Figure 21, we show the sequence diagram for granting a floor to a participant by the application logic. The conference setup is assumed to already exist among the three participants (P1, P2, and P3), participant 1 has requested the floor. The floor control policy is assumed to be first-ask-first-served. The application logic checks the queue and grants the floor to participant 1 based on the queue.

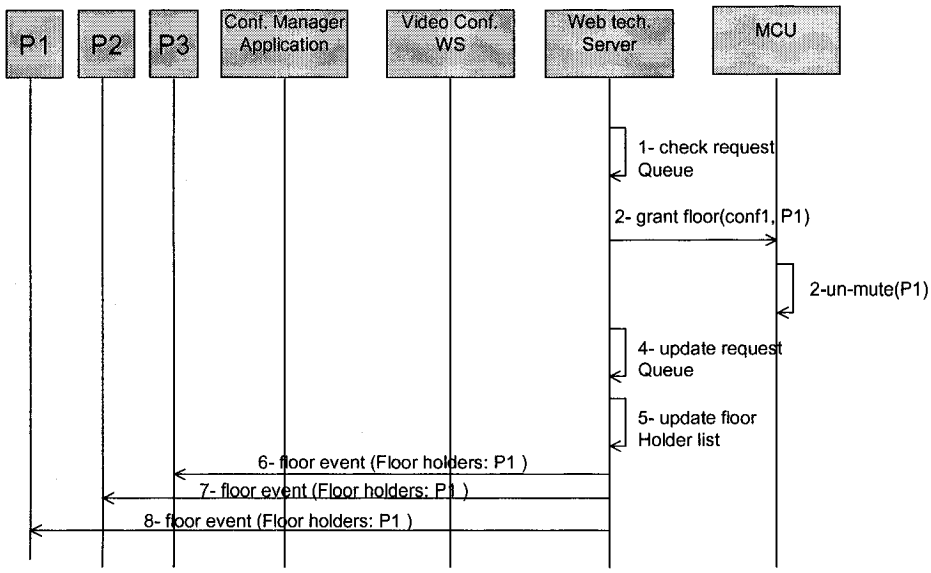

Figure 21: Grant a floor by the application logic

Description of Events:

1. The Web Technology Application Server (implements the advanced video conferencing application WS) checks the floor requests queue.

2. The Web Technology Application Server decides to grant the floor to participant 1 based on the application logic. It maps the action to a PARLAY API call, and sends it to the MCU using CORBA.

3. The MCU, internally, asks the media manager to un-mute participant 1.

4-5. The Web Technology Application Server updates the floor requests and floor holders' lists.

6-8. The Web Technology Application Server notifies all participants with an update to the floor requests and floor holders' lists.

Scenario 4: Revoke a Floor by the application logic

In Figure 22, we show an application logic revoking a floor from a participant. The conference setup is assumed to already exist among the three participants (P1, P2, and P3), and participant 1 holds the floor. The floor control policy is assumed to be a maximum time limit for holding the floor. The application logic checks that the time elapses and decides to revoke the floor from participant 1.

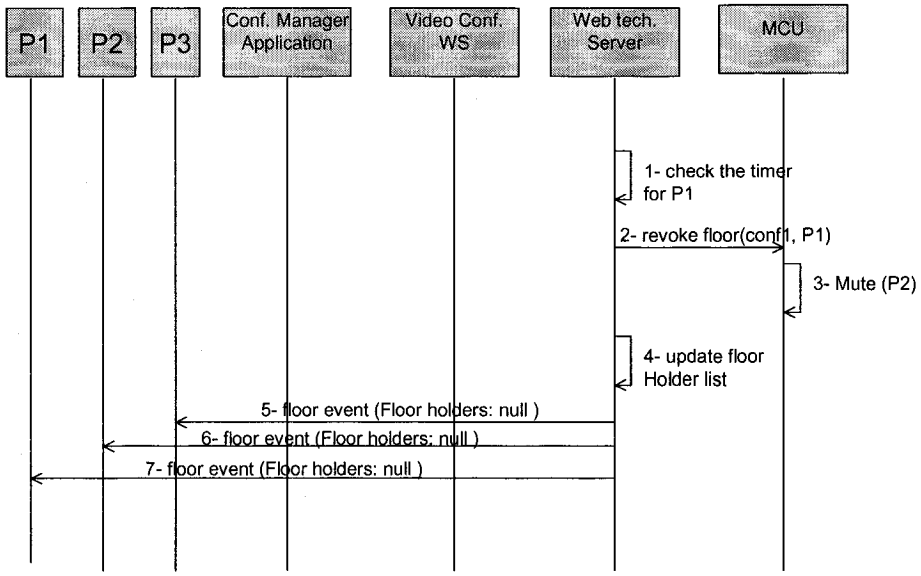

Figure 22: Revoke a floor by the application logic

Description of Events:

1. The Web Technology Application Server (implements the advanced video conferencing application WS) checks the timer.

2. The Web Technology Application Server decides to revoke the floor from participant 1 since the time limit for holding the floor elapses. It maps the action to a PARLAY API call, and sends it to the MCU using CORBA.

3. The MCU, internally, asks the media manager to mute participant 1.

4. The Web Technology Application Server updates the floor holders' list.

5-7. The Web Technology Application Server notifies all participants with an update to the floor holders' list.

Scenario 5: Revoke a Floor by a chairman

In Figure 23, we show the sequence diagram for revoking a floor from a participant by a chairman. The conference setup is assumed to already exist among the three participants (P1, P2, and P3), participant 1 holds the floor, participant 2 has the role of a chairman (or floor manager) and has the rights to revoke the floor to any party holding the floor. The floor control policy is assumed to be chair-controlled.

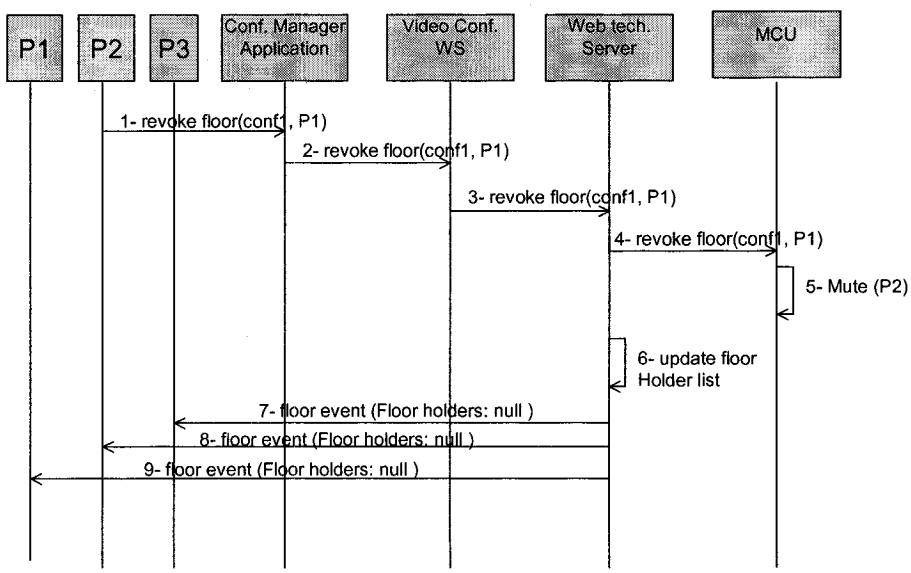

Figure 23: Revoke a floor by a chairman

Description of Events:

1. Participant 2 (chairman) sends an HTTP POST to the Conference/Floor Manager Application to revoke a floor from participant 1.

2. The Conference/Floor Manager Application maps the request to XML format and sends the request to the video conferencing application web service

3. The Video Conferencing Application WS sends the data using SOAP to the Web Technology Application Server.

4. The Web Technology Application Server (implements the advanced video conferencing application WS) analyses the message and decides to revoke the floor from participant 1 based on the chairman's request, maps it to a PARLAY API call, and sends it to the MCU using CORBA.

5. The MCU, internally, asks the media manager to mute participant 1.

6. The Web Technology Application Server updates the floor holders' list.

7-9. The Web Technology Application Server notifies all participants with an update to the floor requests and floor holders' lists.

Scenario 6: Release a Floor by a participant

In Figure 24, we show the interaction among the main components. The conference setup is assumed to already exist among the three participants (P1, P2, and P3). Participant 1 holds the floor.

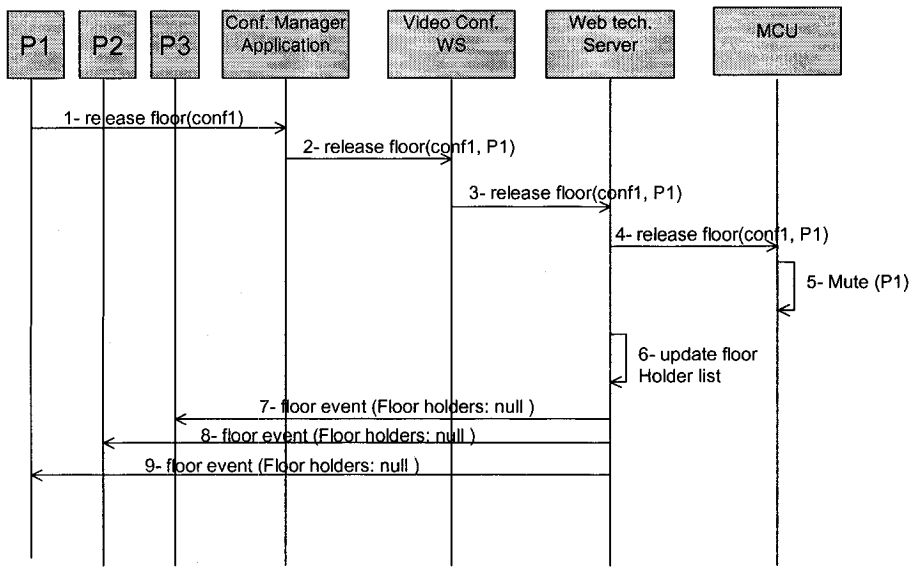

Figure 24: Release a floor by a participant

Description of Events:

1. Participant 1 sends an HTTP POST to the Conference Manager Application to release the floor.

2. The Conference/Floor Manager Application maps the request to XML format and sends the request to the video conferencing application web service.

3. The Video Conferencing Application WS sends the data using SOAP to the Web Technology Application Server.

4. The Web Technology Application Server (implements the advanced video conferencing application WS) analyses the message, and maps it to a PARLAY API call, and sends it to the MCU using CORBA.

5. The MCU, internally, asks the media manager to mute participant 1.

6. The Web Technology Application Server updates the floor holders' list.

7-9. The Web Technology Application Server notifies all participants with an update to the floor requests and floor holders' lists.

4 PARLAY Mapping

Our goal is to identify the PARLAY API methods needed for the interaction between the Web Technology Application Server and the MCU and to find out if the mapping with the proposed extensions (p. 32) fits well in our proposed architecture.

Web Technology Application Server (Web Service interface)	MCU (PARLAY API methods)
requestFloor()	floorRequest()
releaseFloor()	floorReleased()
grantFloor()	appointSpeaker()/inspectVideo()
revokeFloor()	revokeSpeaker()/inspectVideoCancel()

We identify one-to-one mapping between the Web Service interface and the PARLAY API for the requestFloor() and releaseFloor() methods. However, we identify one-to-many mapping between Web Service interface and the PARLAY API for the grantFloor() and revokeFloor() methods. In addition, using the Web Service interface we can support additional methods without the need for a mapping operation to the PARLAY API if it does not need any manipulation for the media streams.

In conclusion, we can say that the use of Web Services for floor control offers more advantages compared to the PARLAY API. It provides a higher level of abstraction than the PARLAY API without the need for telecommunication background. It also enables more flexibility for implementing conferencing applications with less time for development and deployment. In the next chapter, we demonstrate our realizations through prototyping.

CHAPTER 5: IMPLEMENTATION

In the previous chapter, we proposed our solution for floor control in a Web Service framework. We mapped the floor control functionalities exposed by the floor control Web Service into PARLAY call control service. We have learned that the use of Web Services was suitable for providing a higher level of abstraction than the PARLAY API, and enables easy integration with other applications and faster time for development and deployment.

In this chapter, we present the software architecture and the video conferencing application we have built based on the proposed architecture described in Chapter 3 and the use of Web Services expressed in Chapter 4.

1 Software Architecture

The key components in the implementation of the floor control are the web technology application server and the multiparty control unit as shown in Figure 25.

The web technology application server hosts the floor control web service (WS), the WS/parlay glue and the parlay client. The floor control WS provides the interface for the floor control functionalities. The WS/Parlay Glue implements the floor control WS interface and maps the WS methods onto the appropriate PARLAY API calls and sends then to the MCU using CORBA through the PARLAY client. The PARLAY client calls the methods that are

supported by the MCU and implements the call back interfaces that allow the PARLAY client to inform the client about what happens in the network.

The multiparty control unit is made of two units: the conference control unit and the multimedia control unit. The control unit performs signaling and management while the media unit provides media processing functionality. The two units communicate via Megaco/H.245.

The control unit is made of the following software modules: the PARLAY conference handler, the PARLAY SIP glue and the SIP handler. The PARLAY conference handler interacts with the application. It receives the PARLAY requests from the application on the one hand, and sends responses/events to the application on the other hand. It also interacts with the media management module. The PARLAY/SIP glue does the actual mapping and the SIP handler interacts with the SIP stack.

The media unit is made of the media manager module and the media handlers. The media manger module interacts with the PARLAY conference handler module and defines the rules to be applied to each media. There is a media handler for each type of media supported. This handler mixes the media by implementing the rules set by the media manager module.

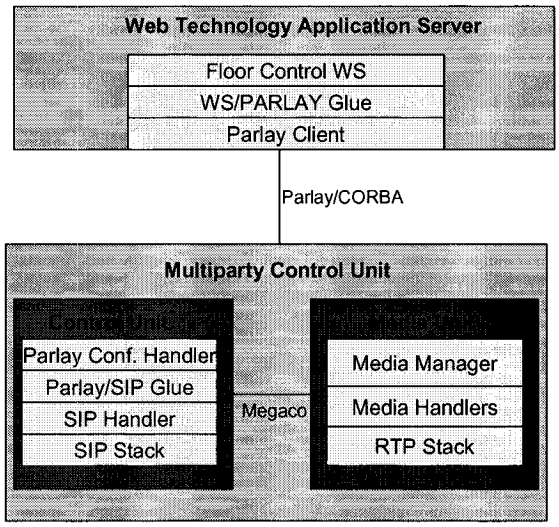

Figure 25: Software Architecture

2 Video Conferencing Application

In this section, we show our video conferencing application. The aim of this application is to show the floor control functionalities extensively in a conferencing scenario based on the proposed architecture we defined in Chapter 3 and the use of the web service framework described in Chapter 4.

The prototype, advanced auto-conducted video conferencing application, combines floor control functionalities with conferencing and presence capabilities into one application. Each action specified by a line or so of XML and transported over SOAP to the Web technology application server. The Web technology server analyses it then maps onto the appropriate

PARLAY API calls and sends to the MCU using CORBA. The MCU executes the call and sends the appropriate requests to Video Conference Clients over SIP. The Video Conference Clients is capable of sending floor control commands over HTTP towards the Conference Manager application.

Figure 26 depicts the overall architecture of the prototype.

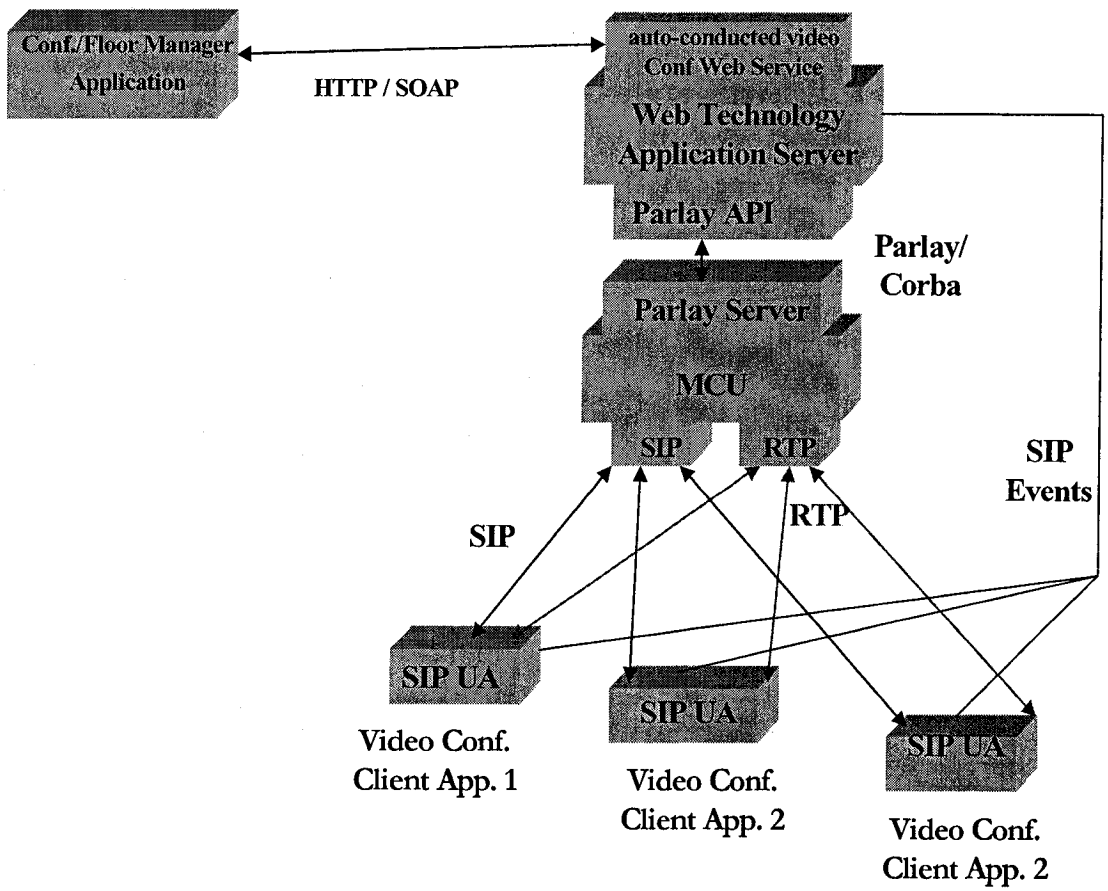

Figure 26: Overall architecture of the Video Conferencing Application

The Main Components

MCU

The MCU acts as a bridge between the Web technology Server and the SIP clients. It is composed of two parts: the Control Unit and Media Unit. The Media Part is the entity that receives, mixes, and distributes all media streams to all participants using RTP. The resources in the media part are controlled by the Control Part entity. The Control Part performs the MCU signaling functionality using SIP. It is responsible for conference call creation, routing, and management. It also gives the applications the ability to manipulate conferences through PARLAY APIs.

Web Technology Server

The Web technology server analyses the messages coming from the auto-conducted video conferencing web service and then maps onto the appropriate PARLAY API calls and sends then to the MCU using CORBA. The video conferencing application web service is implemented and hosted in the Web technology server.

Auto-Conducted Video Conferencing Web Service

The auto-conducted video conferencing web service provides floor control functionalities with conferencing and presence capabilities. Each action in the advanced video conferencing web service is specified by a line or so of XML and sent using SOAP to the Web technology application server.

Conference/Floor Manager Application

The Conference/Floor Manager application as shown in Figure 27 uses the auto-conducted video conferencing web service to implement a conferencing management application that benefits from the network functionalities that are exposed by the Web Service.

Video Conference Client Application

The video conference client application as shown in Figure 28 is built on top of the SINTEL User agent, it shows the floor control functionalities on the client. The moderator and the participant can have access to the floor control functionalities depending on their role. For instance, a moderator can grant or revoke a floor from a participant and a participant can request or release the floor.

3 Graphical User Interface

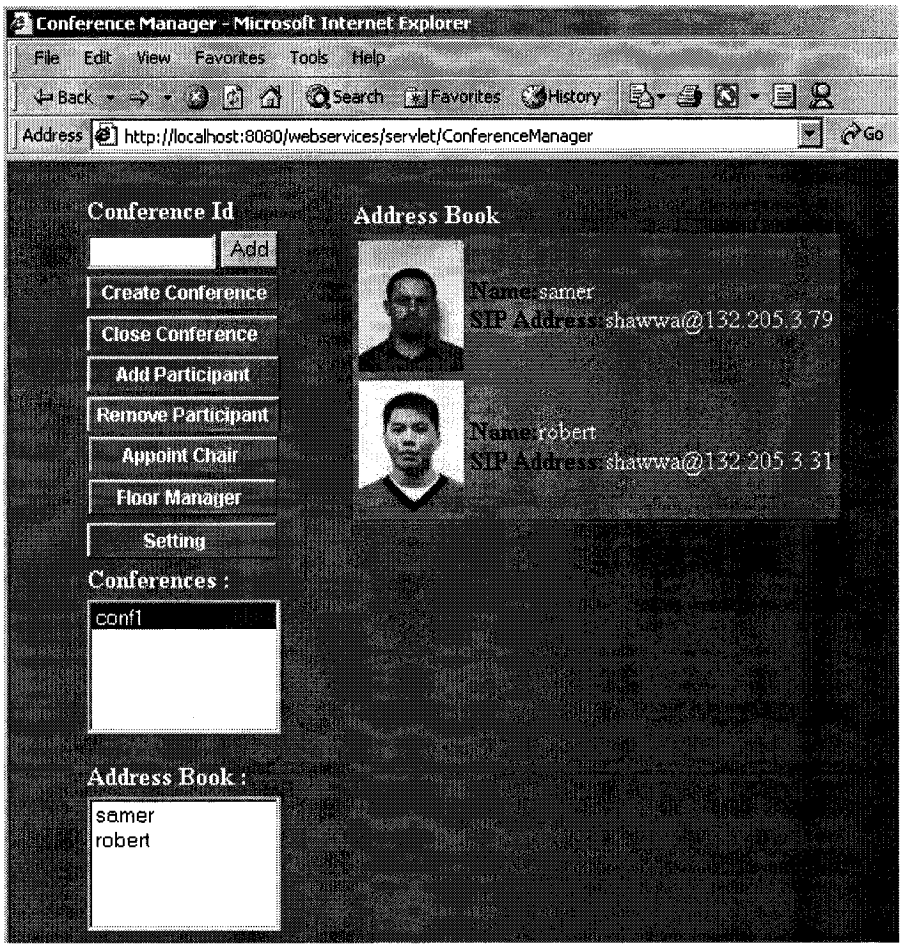

Figure 27: A screen shot of the Conference/Floor Manager Application

First, you must enter a conference name or identifier in the text field then click on Add. This will result in your conference ID being displayed in the Conferences list.

Next, highlight a conference ID from the Conference list, and also highlight the participants you wish to be part of the conference from the Address Book list (you may hold down the shift key to select more than one participant). Then click on "Create Conference".

To end the conference, highlight the conference ID you wish to end, then click on "Close Conference".

To add/remove a participant to a conference, highlight the Conference ID that you wish to add/remove a participant to/from. Then highlight the participant to add/remove. Then click on "Add Participant" / "Remove Participant".

If you want to appoint one of the participants as the chairman or moderator, highlight the conference ID and the participant who is to be the chairperson then click on "Appoint Chair". The Grant and Revoke buttons on that participant's video conference client will now be enabled. Alternatively, the chairperson could be someone other than an active participant. The chairperson can grant and revoke the floor through the web page by highlighting the conference ID and then clicking on "Floor manager".

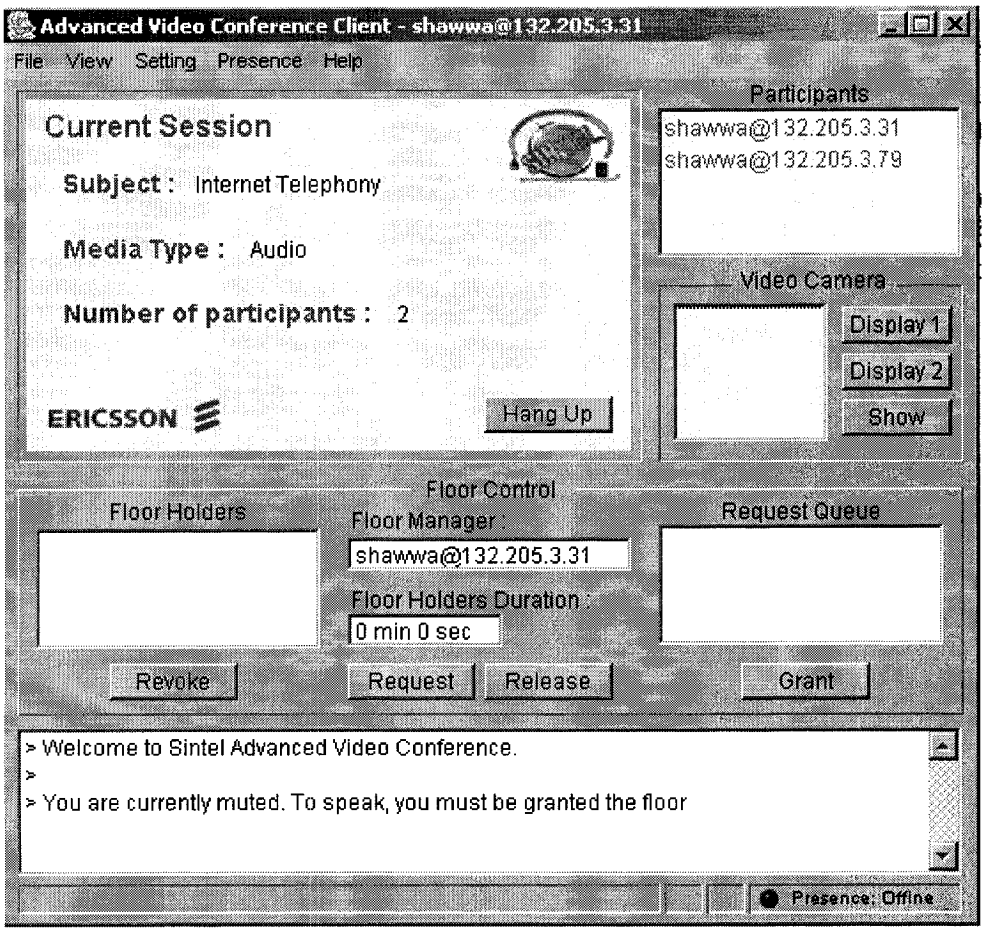

Figure 28: A screen shot of the videoconference client application

If floor control is used, then the clients that are invited are muted by default until the participant is granted the floor, that is, a participant cannot be heard by the other participants of the conference until the participant is granted the floor. Before the participant is granted the floor, the participant must request the floor by clicking on "Request Floor". This will result in the participant being entered into the request queue. The chairperson can select a participant from the request queue and grant that participant the floor by highlighting the participant's

address in the request queue list and then clicking on "Grant". To revoke the floor, the chairperson can highlight a participant in the floor holders' list and then click on "Revoke". More than one participant can hold the floor at one time.

To show video on the video conference clients, highlight the video camera stream you wish to see and then click on either "Display 1" or "Display 2".

4 The Interface

The methods implemented for the conferencing/floor control web service are the following:

- createConference(): This method is used to create a new conference. It takes as input parameters the list of the participants, the floor control policy, e.g., first-ask-first-served or chair-controlled, and the presence quorum to start the conference, and the time duration to hold the floor. It gives as output the identifier of the conference.

- endConference(): This method is used to terminate a conference. It takes as input the identifier of the conference. It gives as output a boolean to indicate if the conference has been successfully terminated or not.

- isConfStarted(): This method is used to check if the conference has started or not. It takes as input the identifier of the conference.

- addParticipant(): This method is used to add a new participant to a conference.

- removeParticipant (): This method is used to remove a participant from a conference.

- appointChair(): This method is used to appoint one of the participants as the chairman or floor manager of the conference.

- getChair(): This method is used to know if there is a chairman for the conference.

- getActiveParticipants(): This method is used to get the list of active participants in the conference.

- getRequestQueue(): This method is used to get the list of participants requesting the floor.

- getFloorHolders(): This method is used to get the list of participants holding the floor.

- grantFloor(): This method is used to assign the floor to one of the participants.

- revokeFloor(): This method is used to force the release of the floor from one of the participants.

- requestFloor(): This method is used to request a floor.

- releaseFloor() This method is used to release a floor.

- subscribeEvents(): This method is used to inform the web service where to send the SIP NOTIFY events to.

- getEvent(): This method is used to get the list of events, e.g., list of the floor holders and list of the floor requests.

5 Tools and Libraries

The prototype is implemented in JAVA. For the media part, we used the JavaSound [36] libraries, which provide low-level support for audio operations such as audio mixing, and audio capturing. As for sending and receiving RTP streams with various audio codecs, we used the JAVA Media Framework [37]

We used WASP Systinet tools [13] for generating the WSDL for Floor control and to generate and send SOAP messages over HTTP. As for deployment, we used the UDDI Service registry from Systinet to register the floor control WSDL description.

CHAPTER 6: CONCLUSIONS

1 Contributions

In this thesis, we defined the requirements of floor control for a generic conferencing application. In addition, we provided an overview of related work in IETF and PARLAY. Based on our evaluation, we proposed new extensions for floor control to the PARLAY API, we described the main components for floor control, and we proposed a new architecture integrating APIs with the protocols.

We have learned that PARLAY APIs and SIP related protocols offer a complete solution for floor control. We demonstrate it by proposing an overall architecture that combines the APIs with the protocols. We believe the use of an API can achieve backward compatibility with older low-level implementations, hide the complexity of the protocol specification, and enable a faster development process.

We have learned that PARLAY as per today's specification is not easy to grasp by developers without circuit switched telephony and IN background. The same developers may also find the level of abstractions to be too low. These two issues motivate us to explore the emerging technologies (i.e., Web Services) as a step further in our study and investigate more on its suitability to offer floor control.

We underline that there was no previous work on the use of Web Services for floor control. Based on our study, we conclude that the use of Web Services is suitable for floor control in

multiparty sessions applications development. A real Web service development environment (i.e., UDDI, WSDL, and SOAP) could be offered. It provides a higher level of abstraction than the PARLAY API, hiding the complexity of the telecommunication networks. It offers more flexibility with application-to-application communication, and faster time for development and deployment. We experienced this through the development of the video conferencing application.

2 Future Work

Today's specification for telecommunication networks focuses on the centralized conferencing schema. An interesting work item could be carried out in future work addressing the floor control issue in a non-centralized conferencing model.

We performed preliminary tests to measure the performance for floor control operations using Web Services. The total time for processing each floor control operation is between tens to hundreds of milliseconds. This means that the time for encoding and decoding SOAP messages is small compared to the total time perceived by the user. More advanced and precise measurements could be tackled in future work to compare the performance of the two solutions for floor control, with or without Web Services.

REFERENCES

1. R. H. Glitho, Advanced Service Architectures for Internet Telephony: A Critical Overview, IEEE Network Magazine, July/August 2000, Vol.14 No.4, pp. 38-44.

2. Third Generation Partnership Project (3GPP), http://www.3gpp.org/

3. M. Handley, H. Schulzinne, E. Schooler, J. Rosenberg, "SIP: Session Initiation Protocol", Request for Comments 2543, Internet Engineering Task Force, March 1999.

4. Internet Engineering Task Force: http://www.ietf.org

5. Simple Object Access Protocol (SOAP), http://www.w3.org/2000/xp

6. X. Wu et al., "Use SIP and SOAP for conference floor control", Internet Draft, Work in Progress, draft-wu-sipping-floor-control-05.txt, November 2003.

7. C. Bormann, SCCP: Simple Conference Control Protocol, Internet Draft (Expired), draft-ietf-mmusic-sccp-02.txt, April 2002.

8. PARLAY forum: http://www.parlay.org

9. S. Desrochers, R. H. Glitho and K. Sylla, Experimenting with PARLAY in a SIP environment: Early Results, IP Telecommunications Services (IPTS) conference, Atlanta, September 2000.

10. R. Glitho, A. Poulin, K. Sylla and O. Cherkaoui, Using PARLAY for Centralized Conferences in SIP Environment, 7th International Conference on Intelligence in Next Generation Networks (ICIN 2001), Bordeaux, July 2001.

11. R. H. Glitho, A. Poulin and F. Khendek, A High Level Service Creation Environment for PARLAY, IEEE International Communications Conference (ICC2002), New York, April 2002.

12. S. Hawwa, F, Khendek and R. Glitho, A Novel Architecture for Conference Floor Control in SIP Environment, Intelligence in Next Generation Networks (ICIN 2003), Bordeaux, France, March 31- April 3, 2003.

13. Systinet http://www.systinet.com

14. H. Schulzrinne and J. Rosenberg, The Session Initiation Protocol: Internet Centric Signaling, IEEE Communications Magazine, October 2000, Vol. 38, No10, pp. 134 – 141.

15. G. Camarillo, SIP Demystified, McGraw-Hill Professional Book Group, 2001. ISBN 0-0713734.

16. M. Handley and V. Jacobson, "SDP: session description protocol," Request for Comments 2327, Internet Engineering Task Force, April 1998.

17. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Request for Comments 1889, Internet Engineering Task Force, January 1996.

18. M. Handley, C. Perkins, E. Whelan, "Session Announcement Protocol", Request for Comments 2974, Internet Engineering Task Force, October 2000.

19. SIP related drafts - http://www.cs.columbia.edu/sip/

20. J. Rosenberg and H. Schulzinne, "Models for multi party conferencing in SIP", Internet Draft (Expired), draft-ietf-sipping-conferencing-models-01.txt, July 2002.

21. S. Hawwa, Audio Mixing for Centralized Conferences in a SIP Environment, IEEE International Conference on Multimedia and Expo (ICME 2002), Lausanne, Switzerland, August 2002.

22. H. Schulzinne, "RTP profile for audio and video conferences with minimal control", Request for Comments 1890, Internet Engineering Task Force, January 1996.

23. A. Roch, "Session Initiation Protocol (SIP)-Specific Event Notification", Request for Comments 3265, Internet Engineering Task Force, June 2002.

24. 3GPP, Open Services Access, Application Programming Interface, Technical specification 29.998, http://www.3gpp.org

25. 3GPP, The Virtual Home Environment, Technical specification 22-21, http://www.3gpp.org

26. UML - The Unified Modelling LanguageUML , http://www.omg.org/uml/

27. CORBA - The Common Object Request Broker Architecture, http://www.omg.org/gettingstarted/corbafaq.htm

28. Distributed Common Object Model (DCOM), http://www.microsoft.com/com/tech/DCOM.asp

29. TINA-C, Business Model and Reference Points, Version 4. 0, http://www. tinac. com/

30. Java Telephony API (JTAPI), Version 1.3., http://java.sun.com/products/jtapi/

31. M. Handley, J. Crowcroft, C. Bormann, J. Ott, "The Internet Multimedia Conferencing Architecture", Internet Draft (Expired), draft-ietf-mmusic-confarch-01, February 2001.

32. ITU-T: Visual Telephone Systems and Equipment for Local Area Networks, ITU-T Recommendation H323, 1995.

33. F. Curbera et al., Unraveling the Web services: An Introduction to SOAP, WSDL and UDDI, IEEE Internet Computing, Vol. 6, No2, March-April 2002, pp. 86-93.

34. Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl.html

35. Universal Description, Discovery, and Integration (UDDI): http://www.uddi.org/

36. Java Sound API : http://java.sun.com/products/java-media/sound/

37. Java Media Framework : http://java.sun.com/products/java-media/jmf/

APPENDIX A (ADVANCED VIDEO CONFERENCING WSDL)

```
<?xml version='1.0'?>
<wsdl:definitions
name='se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'
targetNamespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl
'
    xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
    xmlns:xsd='http://www.w3.org/2001/XMLSchema'
    xmlns:tns='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'
    xmlns:http='http://schemas.xmlsoap.org/wsdl/http/'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:mime='http://schemas.xmlsoap.org/wsdl/mime/'
    xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
    xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'>
    <wsdl:types xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:tns="urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <xsd:schema
targetNamespace="urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl
">
            <xsd:complexType name="ArrayOfstring">
                <xsd:complexContent>
                    <xsd:restriction base="SOAP-ENC:Array">
                        <xsd:sequence>
                            <xsd:element name="item"
type="xsd:string"/>
                        </xsd:sequence>
                        <xsd:attribute ref="SOAP-ENC:arrayType"
                            wsdl:arrayType="xsd:string[]"/>
                    </xsd:restriction>
                </xsd:complexContent>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name='ConfServerWsImpl_appointChair_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_removeParticipant_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getChair_Response'>
        <wsdl:part name='response' type='xsd:string'/>
```

```
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_createConference_Response'>
        <wsdl:part name='response' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_grantFloor_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='tns:ArrayOfstring'/>
    </wsdl:message>
    <wsdl:message
name='ConfServerWsImpl_getActiveParticipants_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getRequestQueue_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_isConfStarted_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getFloorHolders_Response'>
        <wsdl:part name='response' type='tns:ArrayOfstring'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_revokeFloor_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='tns:ArrayOfstring'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_removeParticipant_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_addParticipant_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_revokeFloor_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_grantFloor_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_subscribeEvents_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getEvent_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_releaseFloor_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_isConfStarted_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_addParticipant_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
```

```
    <wsdl:message name='ConfServerWsImpl_endConference_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_releaseFloor_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_endConference_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getChair_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getFloorHolders_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_requestFloor_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getRequestQueue_Response'>
        <wsdl:part name='response' type='tns:ArrayOfstring'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_appointChair_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='xsd:string'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_subscribeEvents_Request'>
        <wsdl:part name='p0' type='xsd:string'/>
        <wsdl:part name='p1' type='xsd:string'/>
        <wsdl:part name='p2' type='xsd:int'/>
    </wsdl:message>
    <wsdl:message
name='ConfServerWsImpl_getActiveParticipants_Response'>
        <wsdl:part name='response' type='tns:ArrayOfstring'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_requestFloor_Response'>
        <wsdl:part name='response' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_createConference_Request'>
        <wsdl:part name='p0' type='tns:ArrayOfstring'/>
        <wsdl:part name='p1' type='xsd:boolean'/>
        <wsdl:part name='p2' type='xsd:int'/>
        <wsdl:part name='p3' type='xsd:boolean'/>
        <wsdl:part name='p4' type='xsd:int'/>
        <wsdl:part name='p5' type='xsd:boolean'/>
        <wsdl:part name='p6' type='xsd:int'/>
        <wsdl:part name='p7' type='xsd:boolean'/>
    </wsdl:message>
    <wsdl:message name='ConfServerWsImpl_getEvent_Response'>
        <wsdl:part name='response' type='tns:ArrayOfstring'/>
    </wsdl:message>
    <wsdl:portType name='ConfServerWsImpl'>
        <wsdl:operation name='revokeFloor' parameterOrder='p0 p1'>
```

```
            <wsdl:input name='revokeFloor'
message='tns:ConfServerWsImpl_revokeFloor_Request'/>
            <wsdl:output name='revokeFloor'
message='tns:ConfServerWsImpl_revokeFloor_Response'/>
        </wsdl:operation>
        <wsdl:operation name='getActiveParticipants'
parameterOrder='p0'>
            <wsdl:input name='getActiveParticipants'
message='tns:ConfServerWsImpl_getActiveParticipants_Request'/>
            <wsdl:output name='getActiveParticipants'
message='tns:ConfServerWsImpl_getActiveParticipants_Response'/>
        </wsdl:operation>
        <wsdl:operation name='subscribeEvents' parameterOrder='p0 p1
p2'>
            <wsdl:input name='subscribeEvents'
message='tns:ConfServerWsImpl_subscribeEvents_Request'/>
            <wsdl:output name='subscribeEvents'
message='tns:ConfServerWsImpl_subscribeEvents_Response'/>
        </wsdl:operation>
        <wsdl:operation name='getRequestQueue' parameterOrder='p0'>
            <wsdl:input name='getRequestQueue'
message='tns:ConfServerWsImpl_getRequestQueue_Request'/>
            <wsdl:output name='getRequestQueue'
message='tns:ConfServerWsImpl_getRequestQueue_Response'/>
        </wsdl:operation>
        <wsdl:operation name='getChair' parameterOrder='p0'>
            <wsdl:input name='getChair'
message='tns:ConfServerWsImpl_getChair_Request'/>
            <wsdl:output name='getChair'
message='tns:ConfServerWsImpl_getChair_Response'/>
        </wsdl:operation>
        <wsdl:operation name='createConference' parameterOrder='p0 p1
p2 p3 p4 p5 p6 p7'>
            <wsdl:input name='createConference'
message='tns:ConfServerWsImpl_createConference_Request'/>
            <wsdl:output name='createConference'
message='tns:ConfServerWsImpl_createConference_Response'/>
        </wsdl:operation>
        <wsdl:operation name='removeParticipant' parameterOrder='p0
p1'>
            <wsdl:input name='removeParticipant'
message='tns:ConfServerWsImpl_removeParticipant_Request'/>
            <wsdl:output name='removeParticipant'
message='tns:ConfServerWsImpl_removeParticipant_Response'/>
        </wsdl:operation>
        <wsdl:operation name='appointChair' parameterOrder='p0 p1'>
            <wsdl:input name='appointChair'
message='tns:ConfServerWsImpl_appointChair_Request'/>
            <wsdl:output name='appointChair'
message='tns:ConfServerWsImpl_appointChair_Response'/>
        </wsdl:operation>
        <wsdl:operation name='getEvent' parameterOrder='p0'>
            <wsdl:input name='getEvent'
message='tns:ConfServerWsImpl_getEvent_Request'/>
```

```
            <wsdl:output name='getEvent'
message='tns:ConfServerWsImpl_getEvent_Response'/>
        </wsdl:operation>
        <wsdl:operation name='requestFloor' parameterOrder='p0 p1'>
            <wsdl:input name='requestFloor'
message='tns:ConfServerWsImpl_requestFloor_Request'/>
            <wsdl:output name='requestFloor'
message='tns:ConfServerWsImpl_requestFloor_Response'/>
        </wsdl:operation>
        <wsdl:operation name='getFloorHolders' parameterOrder='p0'>
            <wsdl:input name='getFloorHolders'
message='tns:ConfServerWsImpl_getFloorHolders_Request'/>
            <wsdl:output name='getFloorHolders'
message='tns:ConfServerWsImpl_getFloorHolders_Response'/>
        </wsdl:operation>
        <wsdl:operation name='isConfStarted' parameterOrder='p0'>
            <wsdl:input name='isConfStarted'
message='tns:ConfServerWsImpl_isConfStarted_Request'/>
            <wsdl:output name='isConfStarted'
message='tns:ConfServerWsImpl_isConfStarted_Response'/>
        </wsdl:operation>
        <wsdl:operation name='grantFloor' parameterOrder='p0 p1'>
            <wsdl:input name='grantFloor'
message='tns:ConfServerWsImpl_grantFloor_Request'/>
            <wsdl:output name='grantFloor'
message='tns:ConfServerWsImpl_grantFloor_Response'/>
        </wsdl:operation>
        <wsdl:operation name='endConference' parameterOrder='p0'>
            <wsdl:input name='endConference'
message='tns:ConfServerWsImpl_endConference_Request'/>
            <wsdl:output name='endConference'
message='tns:ConfServerWsImpl_endConference_Response'/>
        </wsdl:operation>
        <wsdl:operation name='addParticipant' parameterOrder='p0 p1'>
            <wsdl:input name='addParticipant'
message='tns:ConfServerWsImpl_addParticipant_Request'/>
            <wsdl:output name='addParticipant'
message='tns:ConfServerWsImpl_addParticipant_Response'/>
        </wsdl:operation>
        <wsdl:operation name='releaseFloor' parameterOrder='p0 p1'>
            <wsdl:input name='releaseFloor'
message='tns:ConfServerWsImpl_releaseFloor_Request'/>
            <wsdl:output name='releaseFloor'
message='tns:ConfServerWsImpl_releaseFloor_Response'/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name='ConfServerWsImplSOAPBinding0'
type='tns:ConfServerWsImpl'>
        <soap:binding transport='http://schemas.xmlsoap.org/soap/http'
style='rpc'/>
        <wsdl:operation name='revokeFloor'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='revokeFloor'>
```

```
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='revokeFloor'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='getActiveParticipants'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='getActiveParticipants'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='getActiveParticipants'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='subscribeEvents'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='subscribeEvents'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='subscribeEvents'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='getRequestQueue'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='getRequestQueue'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='getRequestQueue'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='getChair'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='getChair'>
```

```
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='getChair'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='createConference'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='createConference'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='createConference'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='removeParticipant'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='removeParticipant'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='removeParticipant'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='appointChair'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='appointChair'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='appointChair'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='requestFloor'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='requestFloor'>
```

```
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='requestFloor'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='getEvent'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='getEvent'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='getEvent'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='getFloorHolders'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='getFloorHolders'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='getFloorHolders'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='grantFloor'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='grantFloor'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='grantFloor'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='isConfStarted'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='isConfStarted'>
```

```
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='isConfStarted'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='addParticipant'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='addParticipant'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='addParticipant'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='endConference'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='endConference'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='endConference'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name='releaseFloor'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='releaseFloor'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:input>
            <wsdl:output name='releaseFloor'>
                <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='urn:se.ericsson.lmc.sintel.confServer.ConfServerWsImpl'/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name='ConfServerWsImpl'>
        <wsdl:port name='ConfServerWsImpl'
binding='tns:ConfServerWsImplSOAPBinding0'>
```

```
            <soap:address
location='http://localhost:6060/ConfServerWsImpl/'/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definition>
```