Final Report

of the Semester Thesis

of

Felix Pfrunder

Configuration of SIP-based Conference Servers

Aufgabenstellung:	Konstantinos Katrinis
Thema:	Configuration of SIP-based Conference Servers
Beginn der Arbeit:	02.05.2003
Abgabetermin:	31.08.2003
Betreuung:	Konstantinos Katrinis, Georgios Parissidis, Prof. Dr. Bernhard Plattner

Abstract (auf deutsch - in german)

Das Aufkommen von neuen Technologien wie das Internet und andere Computernetzwerk Technologien und die immer grösser werdende Bandbreite, ermöglicht es Dienste, die Ort gebunden waren, unabhänging vom Ort anzubieten. Es gibt verschiedene solche Dienste, eine Vorlesung, eine politische Debatte, eine Teamsitzung, eine Pressekonferenz, ein Brainstorming- Meeting oder eine Tagung.

Dienste dieser Art sollten Übertragung von Daten über mehrere Medien unterstützen. Bei einer Vorlesung gibt es zum Beispiel einen Referenten der spricht, eine Wandtafel und eventuell ein Beamer oder Videogerät, das die Vorlesung zusätlich mit Anschaungsmaterial Unterstützt. Bei einer typischen Konfiguration kann jeder dazwischen rufen, die Wandtafel, der Beamer oder das Videogerät wird aber nur vom Referenten bedient. Es gibt also in einer Vorlesung mehr und minder priviligierte Teilnehmer. Diese Privilegien können sich ändern, wenn ein anderer Referent die Vorlesung weiter führt. Eine andere Eigenschaft, die unterstützt werden sollte, ist die Möglichkeit, sich in Gruppen zu unterhalten, ohne dass der Rest der Vorlesung etwas davon mitkriegt. Es kann aber auch sein, dass genau diese Fähigkeit Verboten ist. Typisches Beispiel dazu wäre eine Teamsitzung, bei der es der Chef nicht bevorzugt, dass seine Mitarbeiter hinter seinem Rücken Geschäfte tätigen. Weitere Beispiele von Konferenzart-spezifischen Privilegien erscheinen z.B. in einer Pressekonferenz, wo nur diejenige Teilnehmer Fragen stellen dürfen, die bezahlt haben und der Rest darf nur zuhören, oder bei einer politischen Debatte, bei der über Staatsgeheimnisse referiert wird: in diesem Fall wäre es erwünscht, dass nur bestimmte Personen teilnehmen dürfen.

Es existieren bereits Applikationen, die eine Konferenz über SIP unterstützen. Was noch fehlte war ein Modell, das beschreibt, wie die Rechte von den Teilnehmern verwaltet werden sollen.

In dieser Arbeit wird eine vorlesungsartige Konferenz als Musterfall angenommen und Policen (Privilegien, Profile von Teilnehmern usw.) wurden so implementiert, dass sie diesen spezifischen Fall abdecken. Trotzdem is unser Design genügend generell, dass es bei der Implementation jener Konferenzart angewandt werden kann. Zusätzlich kann unsere Implementation für die Realisierung von Konferenzen anderer Art direkt übernommen werden, indem die eindeutigen Erweiterungsmechanismen gebraucht werden.

Abstract (auf englisch - in english)

The wide spread of new computer network technologies such as the Internet and the steady growth of bandwidth enabled the realization of online services that in the past have been spatially bounded to a place. These can now hopefully be offered independent of any location. Typical examples of such services are a distance learning course, a political debate, a team meeting, a press-conference, or a congress.

Services of this kind should support the transmission of content over different types of media. A referee, who is speaking, a whiteboard and probably a beamer or a video to show additional illustrative material have to be supported to run a lecture. Every participant can interrupt the lecturer by asking questions in a typical configuration of a lecture, but only the moderator can operate on the whiteboard, the beamer or the video. Therefore we differentiate in a conference between privileged and non privileged participants (roles). The assignment of privileges may dynamically change during the conference course, if for example another referee takes over the moderation of the lecture. Another feature that is frequently desired is the ability to form small subgroups for discussion. It should be possible to communicate with all the participants of such a small group, without that the rest of the conference receives the content exchanged in the scope of the subgroup. On the other hand it should be possible for this feature to be prohibited for certain participants. For instance the director of a company might not like his workers having private meeting among them during a company meeting and without him participating in this sub-conferences. Another example, where conference privileges play an important role, is a press-conference, where only a group of registered reporters have the right to ask questions, whereas the rest has only the right to listen.

There exist already applications, that implement conferencing over SIP. Still, a model that describes how policies of participants should are managed in the scope of SIP conferencing is missing.

This thesis asserts a lecture-like conference model as the model case. As such, the policies (privileges, profiles of the participant etc.) have been customized to implement exactly this special case. Nevertheless our design is generalized enough to be easily adaptable to any conferencing scenario. To serve this purpose, unique extension mechanisms have been used.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

In a moderated computer supported conference several participants communicate synchronously towards a common objective using simultaneously numerous media components, as for example audio/video stream, shared presentation tools and shared white boards. An instance of an "umbrella-protocol" termed *conference control protocol* running at the end-system of every participant (and at the conference server in the case of centralized topologies) acts as the gateway of the local user to the conference: it facilitates reception and presentation of events related to conference state and, more importantly, provides conference control services significant to the local user. These services can be categorized as conference creation (including sub conferences in the scope of the main conference), conference join/leave, membership control and floor control. We characterized the protocol as an "umbrella-protocol" due to the fact that each service category is implemented by a stand-alone protocol. Two reasons speak for the use of a modular solution against a monolithic all-inclusive conference control solution: extensibility and exploitation of well-established protocols to realize part of the service categories. The first reason being obvious, we focus on the second.

Conference creation refers to the specification of the details of a conference instance, such as list of users authorized to join the conference, assignment of profiles to admitted users, title, start time and duration of the conference, preferred media components and selection of their configuration, among others. The capability of creation of sidebars - ad hoc sub conferences in the scope of the main conference - is also included in this service class.

Conference join/leave encloses location unaware identification and addressing of users and mechanisms for (re-)negotiation of the configuration parameters of media components used by a conferee.

Membership control comprises of services like inviting users to the conference and ejecting joined participants.

Floor control regulates the number of concurrently active users in moderated conferences. In these scenarios only a predefined maximum number of participants is allowed to interact at any time instance, while the rest is following the input of the active users. In case floor management is human controlled, the controlling entity is called moderator.

In the scope of a specific conference scenario should not every participant have the permission to use all of the commands offered by the formerly listed service categories. Typical cases exemplifying the need for privilege differentiation are:

- only conferees with moderator privileges should have the permission to alter the state of the floor status, e.g. interrupt the current speaker and hand over the right to speak (a logical "floor token") to another participant or specify the current maximum number of concurrent active participants.
- the right to dynamically create a sidebar and specify its initial participant's roster.
- specify the list of users authorized to join the conference instance and the profile assigned to each.
- specify whether a participant has the right to invite other users or to expel joined participants.
- configure the media stream map, which describes who is sending/receiving to/from whom or how many concurrent streams of a single media component will be a participant receiving during the conference (e.g. multiple video streams).

Clearly there appears the need to define participant profiles and specify the access rights of each profile to any of the commands of the service categories. The set of the profiles implemented by a conference scenario and the associated mapping of each profile to access rights constitutes

a *conference policy set*. Abstractly set, the implementation of a conference policy set comprises of

- specification of its syntax/semantic and storage technology (e.g. representation language like XML, storage system like a database management system or plain files).
- specification of a protocol used to retrieve or update information stored in the policy (access primitives and their reflection to specific implementation technologies).
- enforcement of the implemented policy by the conference managing entity (conference server in centralized scenarios or every participant in fully distributed scenarios).

A conference server based on SIP does already exist. This server can handle users that like to join or leave. It offers an API to start or terminate sub-sessions. However, this server supports only conferences where all users have the same rights (flat model).

Our goal was to add an additional service to this conference server to support user management based on the assigned privileges to each user. More over we designed and implemented an application that administers - apart from the policies - all information needed to run a conference, as for example who is the active speaker or what type of video stream is received by each particiapnt.

1.1 Use Cases

Use cases are a very helpful instrument to design an application. They allow validation of a model for the application. The model has to be check with every use case and it is desired that every use case is supported by the model. Of course, many more use cases are supported by the our model and the set presented below gives only an outline of the supported feature:

- User A wants to start a conference with users B, C and D. For this purpose he creates the conference and invites B, C and D to it. B is good in security issues, so A wants him to help editing the conference policy. A, B, C should be able to create sidebars. C should also be able to adjust the volume of each of the audio streams he receives.
- User A can change the policy “Invite” or “Eject” of B and C.
- There are two conference instances running on the server. B can join one of the two.
- User A wants to start a conference, where nobody is able to open a sidebar, because he wants to be fully aware of what is being communicated.
- The technical assistant B can change the volume or the video layout, but he is not allowed to join to the conference.
- User A prohibits B to speak, because he is only allowed to listen to the conference.
- No participant is allowed to speak.
- No participant is allowed to send data to the whiteboard.
- Only one user is allowed to transmit data of a certain media type at any time instance.
- A group of maximum *n* users is allowed to speak concurrently.
- At most two sidebars can be opened in the scope of a single conference.
- User A receives only audio/video and user B only whiteboard and audio data.

2. Terminology

In the scope of SIP conferencing numerous technical terms are used. To clarify the documentation following and to avoid any misunderstanding, we review briefly the terminology used herein. The latter is in compliance with the terminology of the work performed so far in the TIK institute in this field and also with the terminology used in the IETF community.

• **active participant:** a participant that is active in a conference, meaning it sends and receives media streams. An inactive participant, to explain the expression more precise, is a participant that does not send/receive streams, but is registered with the policy server.
• **agent:** the gateway of a simple participant to the conference. It is responsible for exchanging messages with the rest of the conference, i.e responding to messages initiated by the session manager and realizing commands received from the participant.
• **cascaded conferencing:** a mechanism for group communication in which a set of conferences are linked by having their focuses interact in some fashion.
• **conference policy:** the complete set of rules manipulated by the conference server. It includes the membership policy and the media policy. [5]
• **conference or session:** constitutes of a number of participants and one or more moderators, that have agreed upon exchanging multimedia data of certain formats during the lifetime of the conference.[9]
• **floor:** the right to send media data to the other participants.
• **focus:** the focus is a SIP user agent that is addressed by a conference URI and identifies a conference (recall that a conference is a unique instance of a multi-party conversation). The focus maintains a SIP signalling relationship with each participant in the conference. The focus is responsible for ensuring, in some way, that each participant receives the media that make up the conference. The focus also implements conference policies. The focus is a logical role.
• **media session:** a (virtual) channel over which media streams are distributed. One or more media sessions can exist at every instance of time during the conference lifetime and a distinct media session is used for every type of media (audio, video, white board channel).
• **membership policy:** a set of rules manipulated by the conference policy server regarding participation in the conference. These rules include directives on the lifetime of the conference, who can and cannot join the conference, definitions of roles available in the conference and the responsibilities associated with those roles, and policies on who is allowed to request which roles.
• **media policy:** a set of rules manipulated by the conference. The media policy is used by the focus to determine the mixing characteristics for the conference. The media policy includes rules about which participants receive media from which other participants, and the ways in which that media is combined for each participant.
• **mixer:** a mixer receives a set of media streams of the same type and combines them in a type-specific manner, redistributing the result to each participant.
• **policy:** a right that determine, what a administrator, participant is allowed to do, change states to open a sidebars.
• **policy server:** an information server, that manages all information (states and policies) needed to run a conference. It offers an interface to the conference server and to the participants. It is similar to the conference policy server mentioned in [5] with the difference, that the policies are divided in states and policies.
• **session holder or moderator:** conference participant(s), who has privileged access on the conference management data. He is responsible for orchestrating the access to shared conference resources (A/V channel, whiteboard tool) and for administering the conference roster (ex. dropping participants).[9]
•**session manager:** a component, that manages the session. It is responsible for session initiation and session control, floor control and synchronization of the session. In our conference model (centralized) an instance of this module runs at the session holder (or sub session holder in case of a sub session). [9]

•**sidebar:** a sidebar appears to the users within the sidebar as a “conference within the conference”. It is a conversation amongst a subset of the participants to which the remaining participants are not privy.[5]
•**simple participant:** individual that takes part in a conference with basic access rights.[9]
•**state:** an attribute that saves a configuration attribute of the conference.

3. Conference Model

SIP can support several topological models for conferencing. In the “end system mixing” model for example, users are interconnected in a tree topology and each user acts as conference server for the others. Another supported model is the “multicast” model, in which one or more multicast addresses are allocated to the conference and are joined/left by the conference users. The “dial-in” model consists of a conference server that maintains a point to point SIP connection to each user. An “ad-hoc” topology, where two users can spontaneously invite a third person to open a conference is a further possible configuration. From all possible SIP conferencing topologies, we will focus on an alternative of the “Centralized Signalling, Distributed Media” model as presented in [3]. The reason for this lies on the fact that an implementation of this conference model already existed in the TIK institute and as such testing our implementation with an existing conference focus proved easy, as long as we confined to the constraints set by the existing conferencing model. However, we do believe that applying our policy control service to other conferencing models should be seamless and require minor changes/add-ons. It must be noted that whenever we refer hereafter to the term “conference”, we will be assuming a centralized conferencing conforming to the model described below in this chapter.

3.1 Topological Model

The “Centralized Signalling, Distributed Media” model consists of a centralized conference server that maintains a point to point SIP connection to each joined participant. The server is only responsible for signalling and conference state management, i.e. no media mixing is performed at the conference server. Media distribution takes place either via multicast or multi-unicast or combination of them.

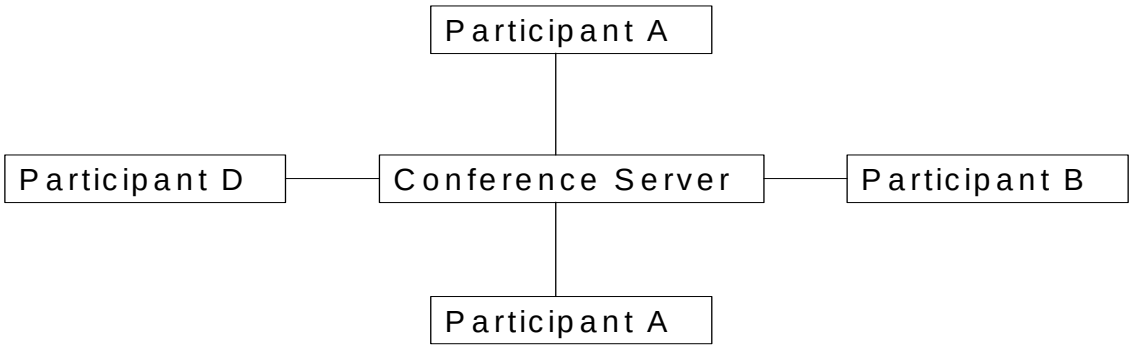

Figure 1: Dial-In Conference Server

The conference is identified by the URI of the conference server’s UA and the participants use this URI to join. For example, if the conference is called *sip:conference21@servers.ch*, the participants have to send an INVITE to the URI *sip:conference21@servers.ch* to join the conference. In case multicast is used to distribute media, the number of participants can scale to large numbers, basically being only limited by the scalability of the underlying multicast routing protocol used. If multi-unicast is used for transmitting media streams, then the conference size de-

pends on the capabilities of the worst participant (assuming that we want to keep a full-mesh model).

3.1.1 Inviting Users to Join

If participant A wishes to ask participant B to join, he uses the SIP REFER message. Participant A sends the following message[1] to B: (HIER: footnote that we don't include the complete SIP messages, but only the headers that are critical for understanding)

```
  REFER     sip:B@example.ch SIP/2.0
From:     sip:A@exmaple.ch
To:       sip:B@example.ch
Refer-To: sip:conference21@servers.ch
```

This causes B to send the following INVITE message to the conference server:

```
INVITE       sip:conference21@servers.ch
From:        sip:B@example.ch
To:          sip:B@example.ch
Referred-By: sip:A@example.ch
```

After the conference server has received this message from B, it performs any necessary authorization steps and starts negotiating media configuration with user B. User B joins then the conference, if he is authorized to do so and a match in media configuration preferences is found.

If participant B does not support the refer mechanism, A can still invite him to the conference by sending a REFER request containing B's URI to the conference server:

```
REFER     sip:conference21@server.com SIP/2.0
From:     sip:A@example.ch
To:       sip:B@example.ch
Refer-To: sip:B@example.ch
```

3.1.2 Users Joining

It is simple for a participant to join the conference, he only needs to send an INVITE message to the conference server. This request message must have its Request-URI set to the URI of the conference server and likewise be addressed (To header) to the conference server. For instance, if B wishes to join *sip:conference21@server.ch*, he should send the following INVITE request:

```
INVITE sip:conference21@servers.ch
From:  sip:B@example.ch
To:    sip:conference21@server.ch
```

3.2 Conference Management

Management of the conferee's roster and floor control, as well as event notification on changes of the conference state are handled by the conference server.

3.2.1 Membership Management

1. Throughout this text we do not include complete SIP messages, but rather only those headers of the SIP message that are critical for understanding the explained feature.

The existing implementation supports two static profiles: conference moderator (or session holder) and simple participant. Each of these profiles has fixed predefined privileges regarding membership control. Specifically, only a moderator entity can invite or expel users. On the contrary simple participants are allowed only to join or leave the conference.

3.2.2 Floor Control

As in the case of membership management, floor control is performed over the conference server through a request/response mechanism. Again here only two static profiles are supported, namely moderator and simple participant. Those differ in some special privileges assigned only to the moderator, like for instance the right to grab the floor or the decision to whom the floor will be passed next.

3.2.3 Subsessions

Subsession or sidebars are managed by the same agent managing the parent session. This agent has the ability to create a new subsession and to invite participants to it. A subsession is created as follows:

- The initiator requests one or more new multicast addresses from the conference focus.
- After the initiator has received the multicast address, he sends an INVITE message to each of the participants he wishes to have a subconference with.
- Every invited participant can accept or deny this invitation.
- If the participant accepts to take part, he joins the new multicast group addresses specified in the INVITE message received from the sidebar initiator.
- Finally, the new sidebar needs to be registered at the conference focus. The latter notifies subsequently the rest of the conference about the newly created sidebar.

The existing model did not contain a mechanism to specify - in advance of the start of a conference or during conference runtime - which participants have the right to initiate a subsession. By default only the moderator possessed this right. Moreover, no limitation in the maximal number of parallel sidebar in the scope of a single conference instance could be specified.

3.3 Conclusion

From the discussion above it should have been made clear that the existing static assignment of rights to only two classes of users (moderator and simple participant) in the existing conference management protocol poses flexibility problems, mainly due to difficulty in the realization of customized conferencing scenarios. To overcome this limitation, we defined a model to handle privileges of conferees on a user granularity, enabling the assignment of custom roles to each user. Additionally, dynamic privilege management during conference runtime is supported.

Besides membership control, the conference server stores and administers media information: it keeps track of what media streams each participant is receiving, the media capabilities of each participant and how the several media streams are being mixed. In what follows we explain thoroughly our model and its interfacing to the existing SIP conferencing infrastructure.

4. Conference Policy Control Model

The policy server uses a single central database to store all information necessary to run a conference. The design of the database was inspired by the UNIX file system that similarly defines distinct classes of users (groups), which are initially assigned a default set of access permissions, and are authorized to perform OS related operations according to these permissions.

Again similar to UNIX systems, we define the "*root user*" profile: a participant with this profile possesses all possible access permissions that a conferee can own.

4.1 Policy server in the SIP - Framework

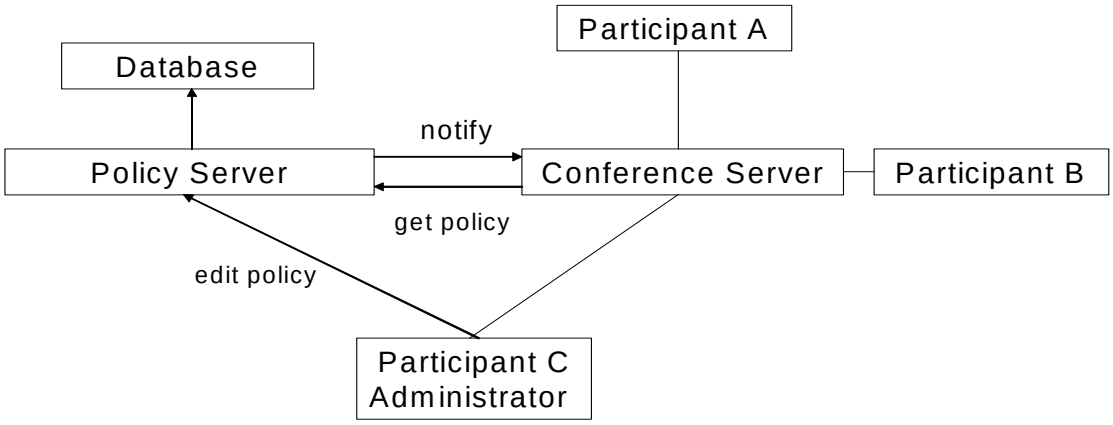

Figure 2: Components in our SIP conference model

In Figure 2 we present the components involved in a SIP-conference, namely the database, the policy server, the conference server and the participant agents. The policy server is responsible for managing the database. It exposes an interface ("*get policy*" interface in Fig. 2) to the conference server that is used by the latter to retrieve all information needed to run a conference. The second interface ("*edit policy"* interface in Fig. 2) offered by the policy server is provided for participants with administrative rights, who intend to change the information stored in the database, to create a new conference or to invite participants. The database comprises of a file that can be accessed and edited by the policy server solely. If certain information is changed the conference needs to be informed, which is done over an interface provided by the conference server. The figure shows three users participating in the conference. Participants C is according to the conference policy the conference administrator. Like every other participant, he keeps a SIP dialog with the conference server notification on changes in the conference status. Additionally he can remotely executes functions on the administrator interface of the policy server to alter the conference policy/state.

4.2 Provide conference status

The conference server should get all information needed to run a conference. The interface ProvideConferenceStatus provides all functionality necessary for that. It provides methods that allow looking up each policy individually. This makes policy management efficient, in case the conference server has to handle a joining participant or a participant trying to invite another individual. Likewise, the interface ProvideConferenceStatus offers for state browsing functionality. This simplifies the work of the conference focus by requiring only a few requests to the policy server to retrieve the necessary information to start the conference.

Every one can look up the conference status if he knows the conference URI. The conference server has to call the browse function of the ProvideConferenceStatus interface that returns the states describing the conference status. This function needs the sip name of the conference as parameter. If the conference pointed with the sip name is not present an exception is thrown.

4.3 Administration interface

The administration interface offers all functionality needed to edit any data stored in the conference database. The policies and states can be read and edited individually, according to the rights of the administrator. If an administrator wants to alter a specific conference state, he has to call the corrersponding function of the administrator interface at the policy server side (Remote Procedure Call paradigm). This function requires the conference identifier, the SIP URI of the administrator and a password for authentication and authorization purposes. Further arguments are a name identifying the state that will be changed and the new value that will be stored for the particular state. If the call succeeds, the function terminates on the client side, otherwise an exception is thrown informing the caller about the reason of the failure.

4.4 Notification of the conference server

Part of the changes lead to reconfiguration of the conference server. Due to the fact that the conference server must be informed about the changes, before he can reconfigure, a conference server interface is exposed by the conference server for this purpose. If reconfiguration at the conference server is needed, the policy server calls (remotely) a function of this interface to inform the conference server accordingly.

4.5 Sidebars

The administrator has to identify himself with his SIP-URI and his password, whenever he wants to open a sidebar. If he owns the required rights, a new conference is created with the administrator as its first participant. The profile, which the first participant is created out of, is called superuser and has enabled nearly all possible rights in this conference. It is not possible to open a new sidebar out of an existing sidebar (no cascading). Due to the fact that a sidebar is always associated with a participant, the root administrator can not create such a sidebar, as he is not a conference participant.

A participant, who wishes to close an active sidebar, must either have the root password of the sidebar or alternatively be a member of the sidebar with the permission to close the sidebar. Internally the right open and close a sidebar is symmetrically implemented, i.e. a participant that has the right to open a sidebar has also the right to terminate the same sidebar.

4.6 Root Session vs. Sidebar Join

The method, which is called when a participant joins a session, must be capable of detecting whether the participant joins a sidebar or the main session. In the case of a participant joining a sidebar, the policy server checks whether the participant has already joined the parent conference and whether he is still an active participant. If not, the handling method signals an error, making it impossible for a conferee to be member of a sidebar without having joined the main session first. As such no authorization procedures are needed during sidebar joins. Similarly if a participant is ejected out of a conference with active sidebars, he is first ejected out of all sidebars he is an active member of.

5. Policy and State Model

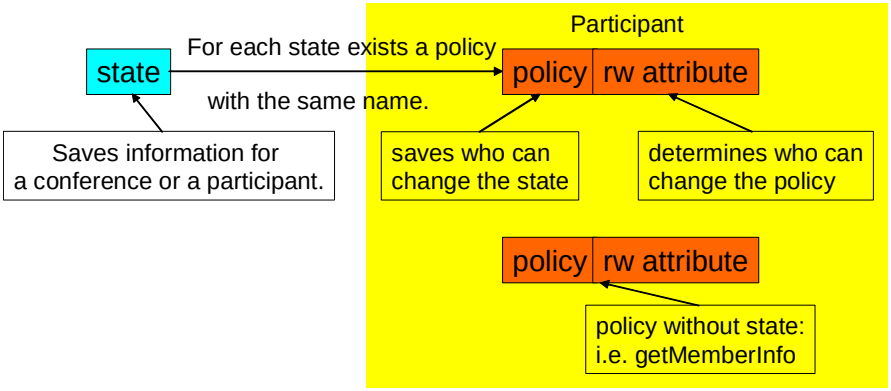

Figure 3: The policy-state model

The policy server stores states and policies. A state holds information about a conference like video resolution received by a participant or conference duration. The set of current values of all states make up the current status of a conference. A policy on the other hand stores information about permissions that participants have, either on performing specific conference management actions (e.g. invite another user to the conference) or altering the conference state (e.g. alter the maximum number of participants allowed).

When a state has been changed by an authorized subject, the conference server needs to be notified in order to apply the changes (e.g. expel a participant). Therefore the policy server is triggered to send a message containing the state change to the conference server, whenever a state change occurs. States are stored either on a conference-wide level, like for example the video layout or the maximum number of participants, or on a participant-specific level, as for instance the volume at which the participants voice stream will be mixed at together with other voice streams.

Contrary to state changes, the conference server does not need to be notified on updates of a policy entry. It is even possible that a policy change is never sensed by a participant. This occurs for example if a participant, who is revoked by the administrator the right to alter any state in a conference, never tries to change a conference state. Rather than being notified, the conference server looks up a policy entry to check whether a participant is authorized to perform a specific operation on conference state. Apparently policy entries refer always to a particular participant, as it is not semantically significant to have conference-wide policies. One reason for conference-wide policies would be to keep the database file small: have a policy that denies or grants a permission to all participants and then single participant policy entries that override the conference-wide entry.

5.1 Connection between states and policies

To clarify the connection between states and policies, the latter are used by the conference server to decide whether a participant is authorized to perform a conference state altering operation. Whenever a participant tries to change a state of another participant or of the conference, the policy server checks whether this is allowed by the participant's policy entry with the same name as the state. If this is the case, the value of the state is set to the new value as specified by the participant's command, otherwise an exception is thrown.

Note that not every policy is assigned a state, whereas a state is always assigned a policy. A policy is assigned a state only if the action that the policy controls permissions on, needs one or more status values stored in the conference database. As an example, the state "lifespan" (full

entry: `<state name="lifespan" value="Mon Jun 2 13:14:25 CEST 2004"/>`) holds the (conference-wide scoped) duration of the conference and can be changed only by participants, who have their corrersponding policy entry “lifespan” enabled (full entry: `<policy name="lifespan" rw="n" use="y"/>`). On the other hand the policy entry *getMemberInfo* that decides whether a participant is allowed to retrace information about the other members of the conference, does not have a corrersponding state entry.

5.2 Controlling of the policy

Changing permissions is controlled by a special attribute (*rw*) contained in each policy entry. If the rw attribute of a policy entry is set to ‘y’, then the participant is allowed to change the every entry of the particular policy of the associated conference, including the one referring to himself. Let us give a short example: if a participant A wants to change the policy entry of participant B, the policy server checks the “rw” attribute associated with the policy of the participant A. Only if this attribute allows that change, the operation is performed. Exceptionally is the “rw” not at checked, if the participant that initiated the operation is authenticated as a user with root permissions.

5.3 Policy and State Aggregations

Policies and states are semantically grouped into aggregations to serve modularity in design and implementation. We list below the main (first-level) policy and state aggregations:

- **partPolicy:** contains all policy entries the permissions of a participant within the conference regarding membership and general conference status management operations, aka not controlling any operations on media stream status control. In particular, the policy entries of this aggregation control permissions on changing state entries contained in the “*partState”* and “*confState”* aggregations discussed later on in this chapter. For each participant there exists a “partPolicy” aggregation. This category constitutes the implementation of the membership policy as described in [5].
- **mediaPolicy:** the “mediaPolicy” aggregation contains atomic entries that refer to specific aspects of a medium used in the conference (currently audio, video and white board), like for instance the volume level of the audio stream or the size of the video stream received by a participant. Policy entries in this aggregation control changes of states contained in the “*partMediaState*” and “*mediaState*” aggregations. There could be numerous “mediaPolicy” instances stored for a single participant. This category implements the “conference media policy” as mentioned in [5].
- **partState:** this aggregation contains all state entries describing the status of a single participant and is therefore saved participant-wide. A change of a state within that aggregation causes the policy server to notify the conference server by remotely calling with the function “partStateChanged”.
- **partMediaState:** states describing the configuration of all media used by a single participant are stored within this aggregation. There exist several “partMediaState” aggregations, with a name indicating the managed media type (audio, video or white board). Whenever a state of this aggregation has changed, the policy server calls remotely the method “partMediaStateChanged”.
- **confState:** groups all states, apart from media stream states, that describe the configuration of a conference. If a state within this aggregation changes, the function “confStateChanged” is called to notify the conference server. This aggregation is saved conference-wide.

• **mediaState:** contains all states controlling the media streams of a conference and has conference-wide scope. The notification of the conference server takes place by calling function “mediaStateChanged”.

The distinction between policies and states is important, because only changes in states need to be reported to the conference server. The second distinction is among states: between the “partState” and “confState” on the one hand and “partMediaState” and “mediaState” on the other hand, depending on whether the states are participant- or conference-wide scoped.

5.4 State Entry Types Specification

A state entry consists of a name identifier and a corresponding value. It applies either to a single participant or to a conference instance. State entries are classified either as (normal) states (including all states besides these that influence media streams) or media states (including all states influencing media streams).

Identifier	Category	Content Type	Description
floor	mediaState	SIP-URI	name of the participant that holds the floor
inputVolume	partMediaState	unsigned integer	the volume a participant speaks
layout	mediaState	an integer value	appearance of the video stream
lifespan	confState	date as string	enddate of the conference
loudestSpeaker	mediaState	SIP-URI	participant that is allowed to speak loudest
maxMixers	confState	unsigned integer	number of available mixers
maxParticipant	confState	unsigned integer	the maximal number of active participants
participant	confState	SIP-URI	name of an active participant, can occur more than once
receive	partMediaState	y or n	determines if the participant gets the media stream type of that category where the receive state is saved

Identifier	Category	Content Type	Description
send	partMediaState	y or n	determines if the participant is allowed to send the media stream type of that category where the receive state is saved
tiled	mediaState	y or n	indicates whether the video is tiled or not

Table 1: The states specification

A state entry in the conference database looks as follows:

```
<state name="loudestSpeaker" value="felix@linux.tik.ethz.ch" />
```

This indicates which user is the one that should be mixed with at the loudest volume. The name attribute stores the string that is used to identify this state through the interfaces. This string has to be passed as parameter, if an administrator likes to edit the state and it is returned as name-value pair string, if the states are browsed by the conference server. The value attribute contains the actual information, which stores the conference state or the participant state and influences the configuration of the conference server.

5.5 Policy Entry Types Specification

A policy entry consists of a name identifying the state, to which the policy rule applies, and of two attributes, “rw” and “use”. Both attributes take values out of the domain {“y”,”n”}, corresponding to yes/no logical semantic. Specifically, the “use” attribute indicates whether a participant is permitted to perform actions that can lead to a change in the status of the state that the particular policy rule is administering (namely *use* the associated state controlling interfaces). Exceptionally, some policy entries do not refer to a particular conference state element, but rather to the permission to read conference state information (e.g. *getMemberInfo* policy) or to perform actions that can implicitly alter the current conference state (e.g. *invite* policy). The “rw” attribute in a policy entry designates whether a participant is allowed to read/change the corresponding policy entry of any of the conference participants (including himself, wherever this makes sense). An identical list of policies is stored for every participant and this list comprises of policies each belonging to one of two categories: “partPolicy” or “mediaPolicy”.

Identifier	Category	Administered State	Description
floor	mediaPolicy	floor	specifies whether the participant can influence floor control state

Identifier	Category	Administered State	Description
getMemeberInfo	partPolicy	-	specifies who can learn to whom he is speaking to, we propose this policy in order to enable Conference-Unaware Participants as mentioned in [5]
inputVolume	mediaPolicy	inputVolume	specifies whether the participant can influence inputVolume state
invite	partPolicy	-	specifies whether the participant can invite another user to the conference
join	partPolicy	-	specifies whether the participant can join the conference
lifespan	partPolicy	lifespan	specifies whether the participant can influence lifespan state
layout	mediaPolicy	layout	specifies whether the participant can influence layout
loudestSpeaker	mediaPolicy	loudestSpeaker	specifies whether the participant can influence loudestSpeaker state
maxMixers	partPolicy	maxMixers	specifies whether the participant can influence maxMixers state
maxParticipants	partPolicy	maxParticipants	specifies whether the participant can influence maxParticipants state
openSideBar	partPolicy	-	specifies whether the participant can open a new sidebar

Identifier	Category	Administered State	Description
receive	mediaPolicy	receive	specifies whether the participant can influence receive state
send	mediaPolicy	send	decides whether the participant can influence the send state
titled	mediaPolicy	titled	specifies whether the participant can influence titled state

Table 2: The policies specification

A policy entry in the conference database looks as follows:

```
<policy name="invite" rw="y" use="y" />
```

This policy specifies whether the participant is allowed to invite users. The string denoting the name of the policy is stored behind the name attribute. Then follows the rw attributed. If this attribute is y then the owner of this policy is enabled to read or change all policies named "invite" of the conference participants. The use attribute specifies if the participant can actually use the policy, in the example above the owner is allowed to invite.

5.6 States and policies with special behaviour

5.6.1 Inviting a participant

If an administrator wants to turn a participant active, which means that the participant is now active listening and speaking in the conference and not just registered with the database, the policy server checks first if the administrator has the right to invite somebody and subsequently checks whether the participant, who will be turned active, is allowed to join. If both conditions are met, a “participant” state with the name of the newly activated participant as its value is added to the database.(e.g. <state name="participant" value="arthur@linux.tik.ethz.ch"/>).

5.6.2 Ejecting a participant by changing his join policy

An active participant is removed, if the use attribute of the “join” policy of this participant is set to “n” value. The policy server deletes all participant states of the expelled participant automatically and a message is sent to the conference server.

Another possibility to implement participant ejection could be done by managing a policy "eject" and providing the method "removeParticipant". This leads to the following situation: if a user is ejected and he is allowed to join (“join” “use” attribute is set to“y”), then he could rejoin, so ejecting a user might have no effect. Another strange situation is, if a participant can change the read write tag of the join policy, but is not allowed to eject a participant. Out of this arise situations in which participants are active members of a conference, but have not the right to join to that conference. This is not a problem, but it is not intuitive, and makes the administration complicated.

5.7 Creating a conference entry

There is a function that creates out of a template an initial conference without any participants. This function requires first authentication with the root password. The initial conference is created with a conference name, which is passed as parameter. The result is a conference with default values stored in its states. The task that follows next could be adding participants and giving them pass phrases. The function provided to perform this operation takes the name of the new participant, a pass phrase and the name of a profile template as parameters. The first participant has to be added by an administrator that has the root pass phrase, but all further participants can be registered by every participant that have their invite policy entry set.

It is not necessary to define all policies, states and participant before the conference starts. The policy server is specially designed to add participant, change states or policies while the conference runs and this is just the normal case.

Apparently it is very easy to invite for example a guest conferee with minimal rights. You first add him based on a template, than change his policies. The most useful would be, you enable his join policy and let him send and receive all kind of media streams, whereas the remaining policies are disabled. The guest participant cannot join without sending a pass phrase to the policy server, but it is possible to define the empty string as pass phrase.

6. Implementation

Figure 4 shows the UML diagram of the policy server. The application code is in the root Java package “confController” and there exists one additional package “confController.element” within the confController package. The lines connecting the several classes mean that each class can instantiate the other. The class “XMLParsed” can instantiate “Conferences” and “Conferences” can instantiate “Conference”. The AdministrationInterface class can instantiate the “SendMessage” class.

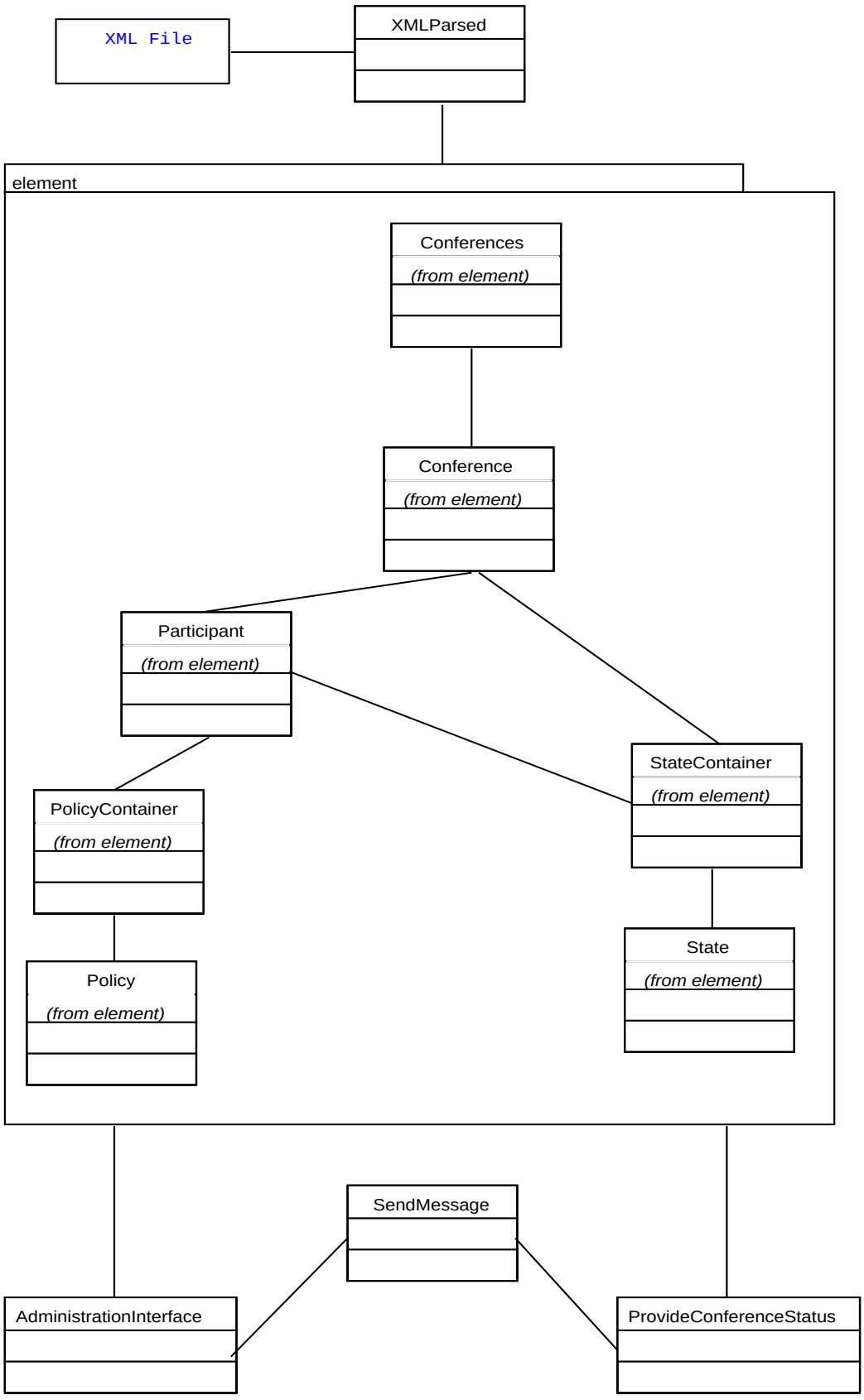

Figure 4:Relationship between the classes of the application

6.1 Database

The database consists of an XML file saved in plain text format. The database the conference server is working on is placed in the file *db/policy.xml*. This file is validated by the corresponding Document Type Definition file *db/policy.dtd* (all paths here are relative to the root installation directory of the policy server code.For more information see Appendix C).

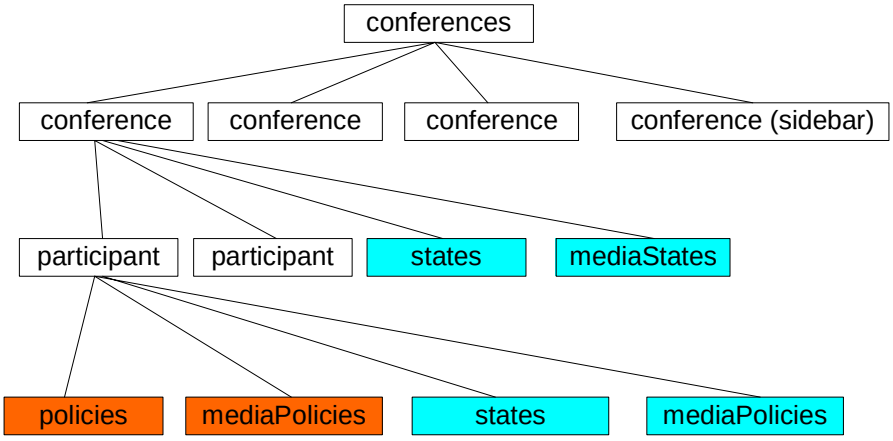

Figure 5: The DOM tree of the XML database implementation

The structure of this database and of an XML database in general is very close to object orientation. It is easy to construct a UML model out of this. We have created for nearly every node a class (conference for example) that provides functionality to create the class of the child nodes.

The policy server application uses Apache Java Xerces to access the elements in the XML file. We preferred using the DOM interface, because it offers an object view of the elements. The alternative SAX interface, which is event based and very convenient, if for example attributes of elements in different hierarchical levels should be edited, did not accomplish our needs as accurately.

6.2 Participant-policy server Interface

If a participant wants to access the policy server directly, then must act as administrator and has to send a password for authentication. The functions implemented in the “AdministrationInterface” class are typically used for that. The class “XMLParsed” that creates the classes out of the confController.element package parses the XML database file to get the required information or to store it. This class provides functions for both: “getConferences()” to look up and change and “save()” to save the changes. If the participant changes a state, then the class “SendMessages” is created, which records the action and causes that the conference server gets informed about the changes. All functions of the AdministrationInterface require the SIP-URI of the administrating user and his password or the root password. If this user simply wants to look up the information stored on the policy server, he utilizes the functions of the “ProvideConferenceStatus”. These functions do not require any authentication.

It was desired, that the conference server and the conference server could be placed on different machines, so they have to communicate over the net. We decided to use a middle ware to simplify the parameter passing. SOAP was chosen due to the fact, that good open source implementations are available. Additional Soap works with HTTP as SIP does.

6.3 Policy server-Database Interface

The “XMLParsed” class parses the XML file and creates a conferences object that is member of the “confController.element” package. The classes of this package read all attributes and its values out of the XML file. These classes are therefore the interface form the database to the policy server. The classes “Conferences”, “Conference”, “Participant”, “Statecontainer”, “Policycontainer”, “State”, “Policy” that help reading, and changing the XML file are similar to the DOM tree of the XML and therefore take advantage of the object orientation of XML.

6.4 Policy Server-Conference Server Interface

The conference server uses the functions of the “ProvideConferenceStatus” class to get the necessary information of the conference. As the “AdministratorInterface” class the “ProvideConferenceStatus” class does create the “XMLParsed” class to parse the xml file. The difference between these two classes is that “ProvideConferenceStatus” does not need to save anything. Due to the browse functionality of the class “ProvideConferenceStatus” the classes “PolicyContainer”, “StateContainer” and “Conference” of the “confController.element” package implement the iterator interface.

The conference server has to offer a service for the notification. The client side of this is implemented in the “SendMessage” class. The name of the function that is called if a change in the database occurs consists of the name of the category where the change happened followed by the string "Changed". If for example the state "maxParticipants" within the category "confState" has been changed then the function "confStateChanged" is executed. The conference server needs to know the name of the conference where the state has been changed, the name of the state, the new value and last but not least it may make a difference whether the state has been added, changed or removed. All these information is passed as parameters to the conference server. If the state that has been changed is a media state then the name of the media type is an additional parameter. The function called if not a state should be change, but for example a new conference should be opened or a new side bar should be added are named like the action that has been performed: “conferenceOpened and sideBarOpenend. The complete list of all function that a conference server should offer is described in the appendix.

6.5 Example

The best way to illustrate the operating mode is to show what happen if the state “maxParticipant” is changed by a participant that has got the necessary rights.

Before we describe how the state is changed it may be useful to learn how the state is read because you probably like to check the value of the state before you overwrite it. The action can be performed by a call over SOAP of the function “browseConfState()” that belongs to the “ProvideConferenceStatus” interface and takes the conference’s SIP-URI as parameter. After that an XMLParsed object is created that in turn creates a “Conferences” object. As next step a “Conference” object is created with the “Conferences” function “getConference()” that takes the conference name (SIP-URI) as parameter. In the following step a “StateContainer” is created with the function “getStates()”. This object implements the iterator pattern that allows passing all states within a conference. A state object is created with the “StateContainer” function “next()” and its name and state are extracted with the state object function “getName()” and “getValue()”. The “NameValuePair” object is filled by its constructor and it is added afterwards into a vector. The commands listed in the last two sentences are repeated until the “StateContainer” function “hasNext()” indicates that no more states remain. Finally a “Vector” is passed back as return value over the SOAP interface. Due to this the administrating user has to search within the “Vector” after the “NameValuePair” object that stores the desired state.

After the user has checked the state he might want to change its value. At this point it should be mentioned that these two actions (reading a state and writing a state) are not in a transaction meaning it is possible that another user has altered the state before our user does this. The function “setConfState()” of the “AdministrationInterface” has to be called over SOAP to change the state. The application allows changes if either the root password is sent as parameter or the participant, that identifies himself with a SIP-URI and a password has got the appropriate rights.

In case that a participant likes to change the state the SIP-URI of the participant and the password have to be checked before anything is done. Nevertheless the database is locked first. The locking is done with the “XMLParsed” object by an explicit call of the function “openTransaction()”. After locking the XML database is parsed and a “Conferences” object that contains the whole content of all configurations of conferences managed by this server is created with the function “getConferences()” of the XMLParsed object. Now the conference that the administrator likes to manage is extracted with a call of the “Conferences” function “getConference()”. This function takes the SIP-URI of the conference. After that, some information about the administrating participant is needed, so a Participant object is created by the method “getParticipant()” of the “Conference” object. The password is checked with the function “correctPassword()” of the “Participant” object. Whether the user has got the necessary rights to administrate is checked next. Out of the “Participant” object a “PolicyContainer” object containing all policies is created. The function “getPolicy()” called with the argument string "maxParticipant" returns an object of a “Policy”. Finally the method “getUse()” of the “Policy” object has to return a string containing "y". If one of these operations fails (wrong password, conference does not exist, participant not present or not enough rights) the database is unlocked and an exception is thrown.

If the root password is sent as parameter the same steps have to be performed as mentioned above till the “Conferences” object is created. The function “correctPassword” of the “Conferences” object checks if the root password is correct, if not the database is unlocked and an exception is thrown.

After the successful authorisation the tag named “maxParticipant” has to be changed. The method “getStates()” of the already instantiated object “Conference” is called and returns a “StateContainer” object. The state “maxParticipant” itself is extracted by calling the “StateContainer” function “getState()” and it is changed with the “State” object command “setState()”. Calling the “XMLParsed” function “save()”, that unlocks the database after saving, so that other changes can be done, saves the changes.

Finally a message to the conference server has to be sent to inform it, that a state has changed. Creating a “SendMessage” object, adding the method name of the “AdministrationInterface” that has been called first with the parameters, does this. The method “send()” of the “SendMessage” object calls the function “confStateChanged()” over SOAP from the conference server so that this server is notified.

7. Issues on Implementations

7.1 Transaction Management

The transaction management that keeps the database in a consistent state, even if a lot of participants are administrating states and policies simultaneously, is implemented in the “XMLParsed” object. It is based on a java “.lock” file that is created at the entry point of each operation that writes back a changed XML database. The function to create a lock is called “openTransaction()” and is executed in the first line of such a subroutine. The file is deleted and the lock is released not until the operation is completed and the file is written back or an excep-

tion occurs. Therefore the function “save()”, which deletes the lock file is executed in the last line of such a function. The “XMLParsed” method unlock, that only deletes the java lock file but does not overwrite the xml database file is call in the catch expressions, that are executed if an exception occurs.

The result of this is that the policy server is blocked even if another call likes to change information that does not affect the information of the blocking operation. Therefore the mechanism implemented is inefficient.

7.2 Profile Templater

The “db/participant.template.xml” saves the profiles or default participants. It is also a complete XML file and can be validated with the corresponding “dtd” file “db/policy.dtd”. The participants are stored with an empty password attribute and with the profile name in the sipName attribute.

Some default conferences, sidebars or profiles for conferences, sidebars are stored in the “db/conference.xml”. The file can also be validated against the “db/policy.dtd”, but there are no participants listed. Instead of a SIP-URI saved in the “conferenceName” attribute, the profile name is saved there and the profiles for sidebars are stored the same as for ordinary conferences, with the difference of "sidebar" written in the owner attribute instead of leaving it empty.

7.3 Adding new policies

It is not possible to add a new policy over the interface calls. It is possible to add a policy by altering the program, to say it more precise, only the database needs to be changed. The policy has to be inserted in every participant into the appropriate category (“partPolicy” or “mediaPolicy”). The actual used database is placed in “db/policy.xml”. If new conferences are opened and new participants are added, the templates have to be updated too. All templates for users are saved in the “participant.template.xml”.

7.4 Adding new states

This is also not possible with an interface method and a bit more complicated as to add a policy, but it suffices also to change the XML files. Due to the fact that for every state must be a policy that controls the state this policy has to be added in the same category as the state. Let me show you an example: If the “confState” "test" should be added, then a policy called "test" has to be added within each participant in the category “partPolicy”. The same has to be done when the state is added as a “partState”. On the other hand if a state within the category “mediaState” or “partMediaState” should be added, the corresponding policy has to be inserted into the “mediaPolicy” with the same name. And of course every participant has to be updated.

7.5 Adding new participant templates

The new participant template has to be added in the template database “db/participant.template. xml”. It has to be complete, I mean for every state in the categories “confState”, “partState” “mediaState” and “partMediaState” must be a policy and the syntax has to be correct, so that it can be validated by the “db/policy.dtd” file. There are no constraints checked by the policy server protecting the application against errors generated by changing any XML files. The name the profile is selected with over the interface has to be written in the attribute “sipName”.

7.6 Adding new conference templates

The file “db/conference.xml” contains all conference templates, so a new conference template has to be added here. In the conference must not be any participants. It has to be possible to validate the new conference template against the “db/policy.dtd” file. The name the conference profile is selected with over the interface method is saved in the conferenceName attribute.

8. Discussion

8.1 Model

In a file system or a database the question occurs, who can edit it and who can edit what. Should a user be possible to either edit all or nothing or should some user edit one part the other another part.

The first model we considered consists of root administrators, that can edit all and users that have got no rights. To make the model really simple there is no connection between the states and the policies as described in chapter 5.1. The advantage of this model is its simplicity. The interface can be reduced (no “rw” policy attribute), the logic in the policy server is simpler and the database smaller, because there are less attributes and no policies for the states. The handling of such a model is easier, because for example it is not possible to take away rights from oneself as in the implemented model, which can be done if the participant changes his rw attribute of a policy.

The reasons why we did not chose that model are, that it does not met the use cases. It is possible to have participants that can join or invite and such that can not, but it is not possible to have participants with different roles, for example a media stream manager is not realizable with that model. For a simple operation as changing the inputVolume for a participant an administrator is needed. The fact, that the first model can be emulated by the second does also militate in favour of the second model.

However, even the complex model cannot handle with every use case. It is not possible to realize, that a participant can manage only the policies of some participants. Such a feature can only be implemented if the name of all participants is stored at each policy. We thought that the cost (disk space, bigger interface) is too high to pale in comparison to the return (the possibility to implement the former use case).

8.2 Interface

The draft [5] proposes that a participant can subscribe by the agent where the conference runs and then get notified if the states or policies change. We decided that it suffices to notify if a state has changed. We like to propose that a subscribed participant gets an interface, where he can learn the policies by calling functions or sending messages over a protocol.

8.3 Implementation

The existing SIP framework has been implemented in Java and a lot of free and useful libraries are also available in java (SOAP an XERCES). Due to these two reasons we decided to take Java as implementation language of the policy server.

The advantage of an XML plain text file is, that it can be read by a developer with a simple editor, which simplifies the debugging of the policy server and later of the conference server. An disadvantage is that we have to implement the transaction management somewhere in the conference server, but the solution based on a java lock file seems to work fine. Of course an SQL database like oracle would be a more efficient and faster solution, particularly if much par-

ticipants like to change information on the policy server simultaneously, but it was not a goal of the problem task to implement a very fast solution.

8.4 Flexibility

The drafts clearly defined some use cases but remain vague with policies and states needed to configure a conference. Especially the media states and policies, which partly depend on the implementation, can not be determined at this stage of development. Due to these two facts we designed the policy server, so that it is very easy to add new policy or state. We succeeded partly. It suffices to change the xml files to add a new policy or state, but this is quite some work because all template files need to be changed either. It may remain writing a script or building an application which does that work.

9. Outlook

Our work shows a possible way of managing the configuration of a policy server. We created an application called policy server, that can manage different users with different rights. They can edit the conference according to their privileges. It can be argued that a conference can be run with a much simpler configuration, but it is also possible to enlarge the whole user management. So we think we have chosen the happy medium.

We decided at the beginning to design the application without considering much about the existing SIP-framework. This lead to a design without any existing waste deposits, but it does not simplify the integration which is outstanding.

The conference does not claim to offer a complete set of policies and states. The use of the policy server will show, if new policies or states have to be added or if some can be removed, how good the choice of the policies or states is.

10. References

[1] J. Rosenberg, G. Camarillo et al. (2002), “SIP: Session Initiation Protocol (RFC 3261)”, Internet Engineering Task Force, June 2002.
[2] J. Rosenberg and H.Schulzrinne, “An Offer/Answer Model with the Session Description Protocol (SDP)”, Internet Engineering Task Force, June 2002.
[3] J. Rosenberg and H Schulzrinne, “Multi Party Conferencing in SIP”, Internet Draft, Internet Engineering Task Force, July 2002.
[4] J. Rosenberg and H. Schulzrinne, “Models for Multi Party Conferencing in SIP”, Internet Draft, Internet Engineering Task Froce, June 2002.
[5] J. Rosenberg, “A Framework for Conferencing with the Session Initiation Protocol”, Internet Engineering Task Force, July 2002.
[6] R. Even, O. Levin et al, “Conferencing media policy requirements”, Internet Engineering Task Force, August 2003.
[7] R. Mahy and N. Ismail, “Media Policy Manipulation in the Conference Policy Control Protocol”, Internet Engineering Task Froce, February 2003.
[8] P. Koskelainen, H. Schulzrinne and Xiaotao Wu, SIP-base Conference Control Framework
[9] R Miethig, M. Ruppen, “Implementation of a protocol for the setup/termination and the management of lecture-like multimedia conferences”, Semester Thesis, TIK ETH Zürich, July

Appendix A - Aufgabestellung

Aufgabenstellung

von der Semesterarbeit

für

Hr. Felix Pfrunder

"Configuration of SIP-based Conference Servers"

Aufgabenstellung:	Konstantinos Katrinis
Beginn der Arbeit:	04.05.2003
Abgabetermin:	10.07.2003
Betreuung:	Konstantinos Katrinis, Georgios Parissidis, Prof. Dr. Bernhard Plattner

1. Introduction

Late advances in the area of information technology, such as the emergence of broadband access networks and the increasing processing capabilities of commodity computer systems enabled the widespread usage of several multimedia applications, such as streaming, conferencing and voice over IP. All these applications require the delivery of services of assured quality from the side of the Application Service Providers and therefore require support of equal quality at the network level. In particular, we are interested in conferencing applications: applications that realize real-time communication among computer users residing in spatially separated sites. Communication is achieved through several types of media streams. The mix of media streams used by a specific conferencing application instance depends strongly on the application target (e.g. a distance learning platform might use a whiteboard tool, whereas for a videoconferencing-based chatroom an audio/video synchronized stream would suffice) and the target infrastructure (e.g. a conferencing application embracing contemporary mobile phones would probably hardly offer a whiteboard component). Apart from the media streams further assets offer communicative awareness and coordination, such as membership information, floor control and cascaded sessions within a conference session.

2. Conceptual Formulation

2.1 Conferencing Signaling using SIP

The Session Initiation Protocol (SIP) [1] is a signaling protocol defined by the Internet Engineering Task Force to be used in applications that require the creation and management of sessions. By the term session we refer to the exchange of data between an association of participants. The functionality offered by SIP can be summarized in the following not exhaustive list:

- user location: determination of the system where an end-user resides
- user capabilities: negotiate the configuration of the media types and media parameters that should be used during the session according to user capabilities.
- session management including dynamic alteration of the session setup (media stream configuration, change in user location) and invocation of services.

As it has been already described in [3], SIP can facilitate the setup and termination of multiparty conferences. Several topological models can be realized, however we will focus on the "dial-in" model as presented in [3]. An instance of such a model is shown in Figure 1. The conference is managed by a single entity, the conference server. The latter maintains a point-to-point SIP session with every joined participant until the participant leaves the conference. Therefore this conference topology is often referred to as "*tightly coupled*". A user that wishes to join the conference should first establish a session (called dialog in SIP terminology) with the conference server. The potential participant communicates over this session her network address (e.g. IP address/port number pair) to the server entity. Furthemore media types (e.g. video codecs) and parameters (e.g. bit rate of audio codec) can be negotiated during the session setup phase. This is achieved through an offer/answer model well defined in the scope of the Session Initiation Protocol [1], [5]. The established SIP dialog is kept alive during the lifetime of the participation of a site in the conference and should be used whenever any of the communication peers needs to alter the configuration of a session related to the conference: for

instance when an end system is assigned a new IP address or when the server needs to signal the participant the creation of a new multicast group for a new video stream. Accordingly, whenever a participant wishes to leave the conference or the server decides to expell a conference member, the established dialog is used to tear-down the session between the conference server and the particular participant.

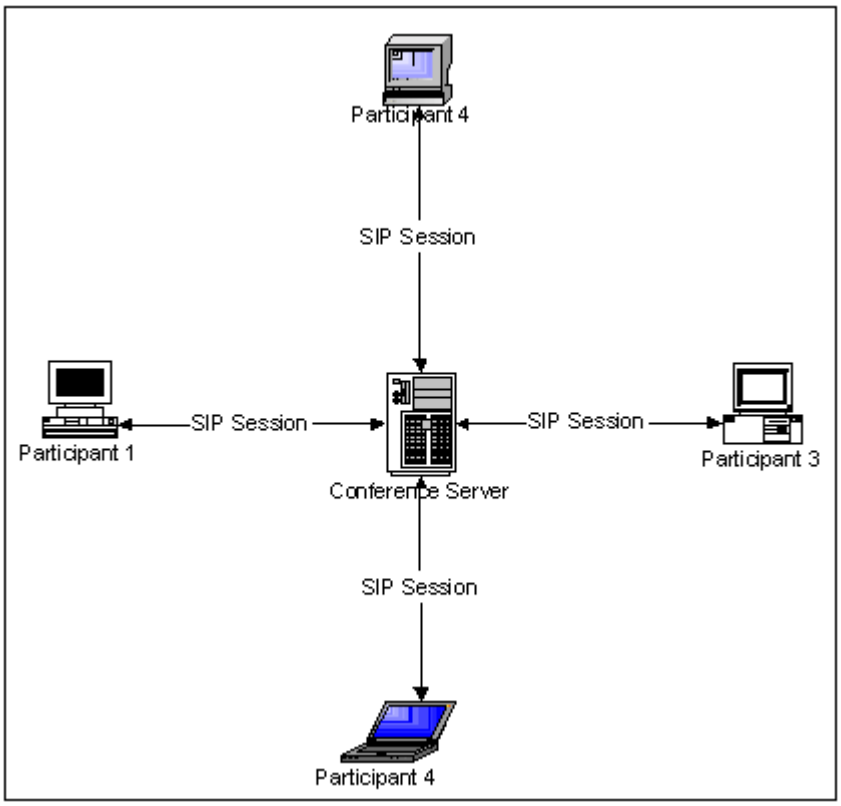

Figure - 1 : SignalingTopology in the centralized conference model

Similar to the SIP topology for conference creation/termination, conference management can be accomodated via SIP means: management of participants (using third party INVITE, REFER and BYE methods), dynamic modification of media parameters and notification about conference state changes [4]. Additionally, media connectivity is handled by the conference server: the server decides about the connectivity graph of media flows streams among participants by configuring the media components of the end systems and potential network media elements (mixers, transcoders, replicators etc.) accordingly.

2.2 Conference Mechanisms vs. Conference Policies

In the former chapter we briefly discussed the functional specification of a conference server with regard to conference creation and management using SIP mechanisms. However, the sound specification of a fully operational conference server requires the determination of a concrete **policy** that is to be implemented via the prescribed mechanisms. As an example the policy table of a conference instance would list the identifiers of the users that are authorized to join the particular conference or likewise contain a role (e.g. listener, simple participant,moderator) assigned to every potential participant according to a predefined participation level scheme.

2.3 Conference Policy

Following logical analysis procedures we structure conference policy in three distinct domains: **participation policy**, **floor control policy** and **media policy**. This is partially compliant with the related IETF RFCs [5], [6], [7] about conference policies, as floor control is not considered as a policy domain in the latter document.

Participation policy should normally contain the list of user identifiers that are authorized to participate in the conference compiled as a record list. Each participant record could additionally contain information about the rights of a participant to perform specific actions that augment the conference membership, such as the right to invite/expell users or to decide whether a potential participant not listed in the starting roster should be admitted to the conference. This can be achieved by defining for example access groups, similar to the UNIX operating systems approach. Each group will by default have predefined rights with regard to membership actions. Evidently the conference server should enforce access groups by performing access control prior to each action and by forwarding specific requests to the members of the appropriate group. For instance consider defining a group of moderators and assigning it the right to decide whether a newcomer, whose identifier is not listed in the list of authorized users to join conference, should be granted or denied admission. The conference server should then enforce the participation policy by forwarding the participant's join intention to the group of moderators.

Floor control policy refers to the interaction control protocol that should be used together with configuration parameters, e.g. human-mediated scheduled floor control, where floor requests are queued in the conference server and floor decisions are taken by a human entity. Apparently the floor control protocol of a policy instance should be chosen out of a pool of protocols that are supported by the conference server. Moreover, the dynamic alteration of the active floor control policy should be enabled, reflecting the intention to use multiple interaction control schemes throughout a single conference instance.

The media connectivity graph is specified in the scope of the media policy. As the term implies the media policy should describe the media connections among the conference participants and any additional processing that should be applied on particular media streams. For example the lecturer in an educational conference instance might wish to receive the video streams from all participants tiled in a single video frame during the whole conference.

2.4 Architectural Considerations

A SIP conferencing architecture needs to be enhanced with additional communication and storage elements to enable full exploitation of the conference policy concept. A policy repository (e.g. a database system) could be used to store the policy instances corrersponding to conference instances. Interaction of a remote client with the policy repository will be provided via a *conference policy server*. The latter will serve client requests to perform read/write/update commands to the repository in a transactional manner. A dedicated protocol between the client and the conference policy server - termed *conference policy control protocol* [5] (abbreviated to CPCP) - needs to be defined for this purpose. Furthermore the conference server has to be extended with appropriate interfaces in order to retrieve information (also remotely) from the policy repository and to be notified when policy update transactions occur. An illustrative representation of the briefly described architecture can be viewed in Figure 2. The components (protocols, elements, interfaces) drawn with dashed lines are the enhancements that are to be researched and implemented in the scope of this semester thesis. Note that the preceding system specification aims to be functional and not to point to specific implementation solutions. For instance, the conference policy control protocol should not necessarily be drawn up as a text based message exchanging protocol. Instead, a Remote Procedure Call Protocol or server side scripting could be deployed (as an illustrative example). The research of implementation alternatives and the justification of the approach adopted constitutes also a task in the scope of

this thesis.

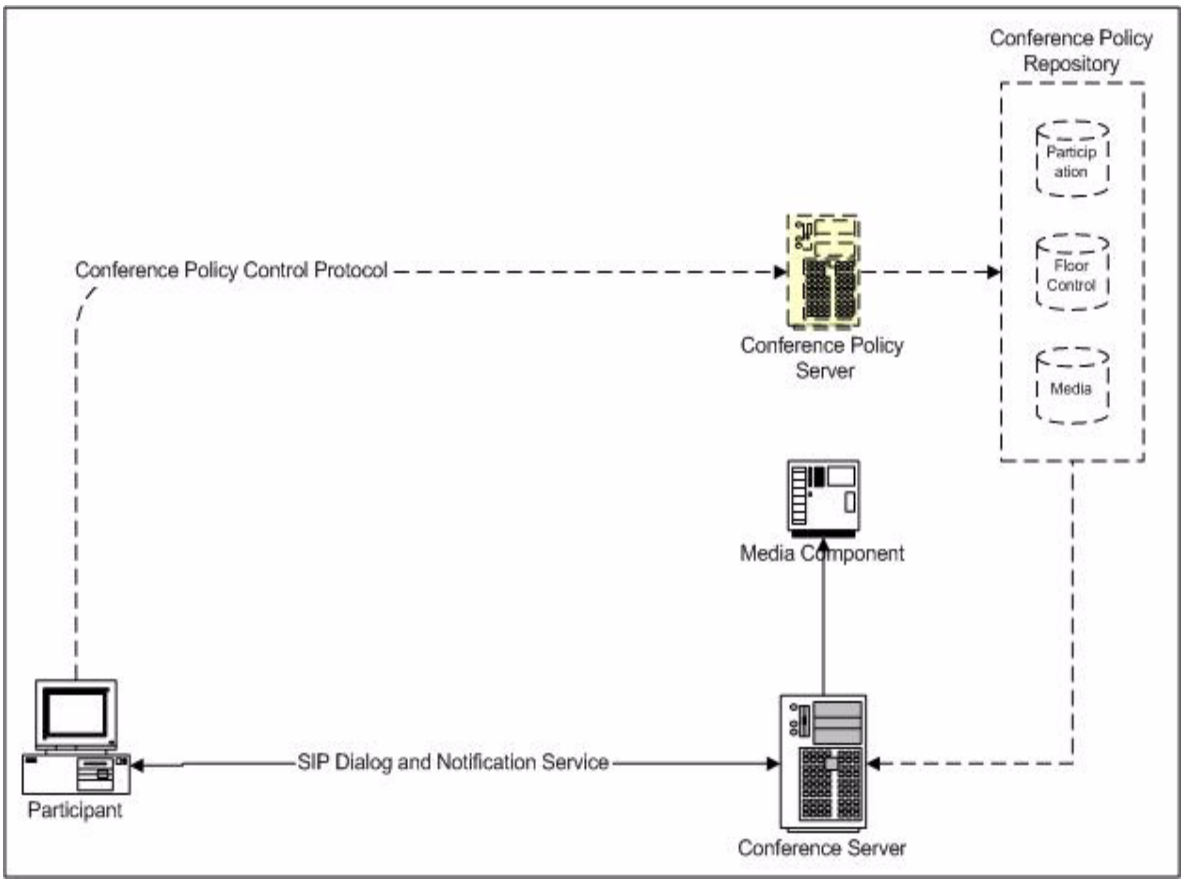

Figure - 2 : SIP Conferencing Architecture enhanced with policy support

3. Procedure

In this chapter a sequence of tasks is listed that should be completed during this thesis:

• Study related literature starting with the references given in chapter 6. These references constitute the core documentation in the subject area and do not claim to be exhaustive. Additional research might prove necessary as the thesis progresses, if new queries arise during detailed specification or implementation approaches (languages, frameworks etc.) need to be considered.

• Formally define the conference policy concept using an appropriate description modelling language. The attributes of a policy instance are not intended to cover every possible type of conference, as it would be difficult to include every possible aspect, but rather reflect the basic set of attributes of a generic conference instance. The collection of policy instances should be stored in a policy repository.

• Specify formally a server entity (conference policy server) that should accept read/write requests by a client destined to the repository. The interaction between client and server will be accommodated through a special purpose conference policy control protocol. It should be also taken into consideration that the communication between policy repository and policy control server might be remote and local communication should not be taken for granted.

• Likewise define the interfaces (remote and local) between the conference server and the policy repository used by the server to retrieve policy information about a specific conference instance.

• Implement the logical design using appropriately chosen software solutions. The preferred approaches should be satisfactorily justified in case multiple solutions are possible.

• Setup a testbed to demonstrate the developed prototype.

4. Important Remarks

• The operating system used for development can be freely selected. For the testbed it is desirable to use several OSes for the participants of the conference.

• A timeplan for the realization of the semester thesis should be sketched by the end of the first week and discussed with the supervisor.

• By the end of the thesis you should compose a written report in the form of documentation of the thesis. The report should be **written in english** understable by a non-specialist. Additionally it should contain all the requirements, design decisions and architectural details.

• The thesis will abstractly be layered into five parts: Literature Study, Requirements Specification, Design, Implementation (including testing) and Documentation. It is advisable to document each phase at its completion time rather than writing the documentation at once before thesis completion.

• In the midterm of the thesis a short intermediate report should be composed and reviewed during a meeting with the supervisors. The intermediate report should list the already achieved tasks and the tasks that are foreseen to have been accomplished by the end of the thesis. Strictly speaking the intermediate report should comply with a structural design of the final report (in a bulleted form).

• An ordinary contact (at least once a week) between the student and their supervisor has to take place via telephone, e-mail, meeting sessions or other means. During these contacts the progress of the performed work has to be presented and problems should be discussed. Especially vital is the daily reading of e-mails.

5. Thesis Results

A fifteen minutes presentation should be given in TIK Institute. The exact date of the presentation will be specified late in the summer semester. Apart from this presentation, the following documents should be handed in on thesis completion:

• A detailed technical report ("Bericht") in english. The following topics should be thoroughly addressed in this report: a description of the investigated research area, a description of the examined design alternatives together with a detailed reasoning in favor of the finally selected design approach, a listing of the solved and unsolved problems (together with the reasons why they haven’t been solved), references to literature, table of contents/ figures/tables and potential appendices (glossary, programming code, state diagrams, call flow examples etc.). The report should end up with an evaluation of how far the initial tasks of the thesis have been achieved and whether the initial timeplan was fulfilled. Five

copies of the final report should be handed in, all bound and double sided printed. The technical report should be composed using Framemaker.

• An abstract in both *german and english*, 1-2 pages long included as a preface in the technical report. This should contain a quick overview of the performed work. The structure of the abstract should be in the form: (1) Introduction, (2) Aims & Goals, (3) Results, (4) Future Work.

• An electronic version of the technical report as well as of all the produced documents (code documentation, models etc.). Figures contained in the final report have to be educationally stored as independent data in a custom-selected format (ex. EPS). The material in electronic form should be either stored on a CD or in a separate directory on an institute's server (accounts should then be created for the students).

• Referenced and processed literature, whether in electronic or printed form.

• A handbook of the implemented system that should contain: system overview, description of the implementation (structure), documentation on data structures and description of test programs. Moreover installation guidelines and potential hardware/software requirements should be included.

• The complete source code of the system and of the test codes, together with all the necessary libraries/APIs/external programs should be handed in. Respectively for the system executables and the test programs.

6. Bibliographical References

[1] J. Rosenberg, G. Camarillo et al. (2002), "SIP: Session Initiation Protocol (RFC 3261)", Internet Engineering Task Force, June 2002.

[2] J. Rosenberg and H. Schulzrinne, "An Offer/Answer Model with the Session Description Protocol (SDP)", Internet Engineering Task Force, June 2002.

[3] J. Rosenberg and H. Schulzrinne, "Models for Multi Party Conferencing in SIP", Internet Draft, Internet Engineering Task Force, July 2002.

[4] J. Rosenberg and H. Schulzrinne, "A Session Initiation Protocol (SIP) Event Package for Conference State", Internet Engineering Task Force, June 2002.

[5] J. Rosenberg, "A Framework for Conferencing with the Session Initiation Protocol", Internet Engineering Task Force, February 2003.

[6] R. Even, O. Levin et al., "Conferencing media policy requirements", Internet Engineering Task Force, August 2003

[7] R. Mahy and N. Ismail, "Media Policy Manipulation in the Conference Policy Control Protocol", Internet Engineering Task Force, February 2003.

Appendix B - Interface Signatures

ProvideConferenceStatus Interface

getPartPolicy
public java.lang.String getPartPolicy(java.lang.String confName,
java.lang.String partName,
java.lang.String policyName)
throws java.lang.Exception

Discovers the policy as called in the param policyName that is in force at the conference confName for the participant partName.

Parameters:
confName - A name of a conference
partName - A name of a participant
policyName - A name of a policy
Returns:
A string containing the use of the policy
Throws:
DBReadException
PartNotFoundException - If the participant has not been found
ConferenceNotFoundException - If the conference is not present
PolicyNotFoundException - If the policy is not present
java.lang.Exception

---

getMediaPolicy
public java.lang.String getMediaPolicy(java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName,
java.lang.String policyName)
throws java.lang.Exception

Discovers the policy as called in the param policyName that is in force at the conference confName for the participant pratName It influences the media type named in the mediaName argument

Parameters:
confName - A name of a conference
partName - A name of a participant
mediaName - A name of a media type (audio, video, whiteboard)
policyName - A name of a policy
Returns:
A string containing the use of the policy
Throws:
DBReadException
PartNotFoundException - If the participant has not been found
ConferenceNotFoundException - If the conference is not present

MediaPolicyNotFoudnException - If the media policy is not present
PolicyNotFoundException - If the policy is not present
java.lang.Exception

---

getPartState
public java.lang.String getPartState(java.lang.String confName,
java.lang.String partName,
java.lang.String stateName)
throws java.lang.Exception

Discovers a state that is in force at the conference with the name confName

Parameters:
confName - A name of a conference
partName - A name of a participant
stateName - A name of a state
Returns:
A string containing the value of the state
Throws:
DBReadException
PartNotFoundException - If the participant has not been found
ConferenceNotFoundException - If the conference is not present
StateNotFoundException - If the state does not exist
java.lang.Exception

---

getPartMediaState
public java.lang.String getPartMediaState(java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName,
java.lang.String stateName)
throws java.lang.Exception

Discovers a state that is in force at the conference with the name confName It concerns the media of the type mediaName

Parameters:
confName - A name of a conference
partName - A name of a participant
mediaName - A name of a media type (audio, video, whiteboard)
Returns:
A string containing the value of the state
Throws:
DBReadException
PartNotFoundException - If the participant has not been found
ConferenceNotFoundException - If the conference is not present
MediaStateNotFoundException - If the media type is not present

StateNotFoundException - If the state does not exist
java.lang.Exception

---

browseConfState
public java.util.Vector browseConfState(java.lang.String confName)
throws java.lang.Exception

Discovers all state of a conference (the active participants belong also to the state

Parameters:
confName - A name of a conference
Returns:
A Vector containing the name of the state and its value
java.lang.Exception

---

browseConfMediaState
public java.util.Vector browseConfMediaState(java.lang.String confName,
java.lang.String mediaName)
throws java.lang.Exception

Discovers all states of a conference for a specific media type

Parameters:
confName - A name of a conference
mediaName - A name of a media type either audio, video, whiteboard
Returns:
A Vector containing the name of the state and its value. Stored as NameValuePair.
java.lang.Exception

---

browseParticipants
public java.util.Vector browseParticipants(java.lang.String confName)
throws java.lang.Exception

Discovers all registered users in a conference no matter if they are active or passive

Parameters:
confName - A name of a conference
Returns:
A Vector containing all names of participants
java.lang.Exception

---

browsePolicy

public java.util.Vector browsePolicy(java.lang.String confName,
java.lang.String partName)
throws java.lang.Exception

Discovers all policies of a single participant

Parameters:
confName - The name of the conference
partName - The name of the participant
Returns:
A Vector containing the name of the policy and the use tag as NameValuePair
java.lang.Exception

---

browseMediaPolicy
public java.util.Vector browseMediaPolicy(java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName)
throws java.lang.Exception

Discovers all policies of a certain media type within a single participant

Parameters:
confName - The name of the conference
partName - The Name of the participant
mediaName - The name of the media type
Returns:
A Vector containing the name of the policy and the use tag as NameValuePair
Throws:
MediaPolicyNotFoundException - If the media of the named type does not exist
java.lang.Exception

---

browsePartState
public java.util.Vector browsePartState(java.lang.String confName,
java.lang.String partName)
throws java.lang.Exception

Discovers all states of a participant

Parameters:
confName - The name of the conference
partName - The Name of the participant
Returns:
A Vector containing the name of the state and the value as NameValuePair
java.lang.Exception

---

browsePartMediaState
public java.util.Vector browsePartMediaState(java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName)
throws java.lang.Exception

Discovers all states of a participant within a media type

Parameters:
confName - The name of the conference
partName - The Name of the participant
mediaName - The Name of the media type
Returns:
A Vector containing the name of the state and the value as NameValuePair
Throws:
MediaStateNotFoundException - If the media type is not present
java.lang.Exception

AdministrationInterface Interface

setUsePartPolicy
public void setUsePartPolicy(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String policyName,
java.lang.String use)
throws java.lang.Exception

Sets the use tag of a policy of an participant in a conference. If the policy join is set from y to n then the participant state of the participant named partName is removed, and the conference server method confStateChanged is executed so the conference server is notified.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - a name of a participant
policyName - a name of a policy
use - a string containing the use y or n
Throws:
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be edited or the administrator has not been found
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information

OutOfRangeException - if the use argument is not y or n
PolicyNotFoundException - if the policy is not present
java.lang.Exception

---

setUseMediaPolicy
public void setUseMediaPolicy(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName,
java.lang.String policyName,
java.lang.String use)
throws java.lang.Exception

Sets the use tag of a media policy of a participant in a conference.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - a name of a participant
mediaName - the name of the media type the policy is within.
policyName - a name of a policy
use - a string containing the use y or n
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information
OutOfRangeException - if the use argument is not y or n
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be edited or the administrator has not been found
MediaPolicyNotFoundException - if the media type is not available
PolicyNotFoundException - if the policy is not present
java.lang.Exception

---

getRwPartPolicy
public java.lang.String getRwPartPolicy(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String policyName)
throws java.lang.Exception

Returns the RW tag of a policy of a participant in a conference, if the asking participant has the right to get it.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - a name of a participant
policyName - a name of a policy
Returns:
a string containing the rw tag
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be edited or the administrator has not been found
PolicyNotFoundException - if the policy is not present
java.lang.Exception

---

getRwMediaPolicy
public java.lang.String getRwMediaPolicy(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName,
java.lang.String policyName)
throws java.lang.Exception

Returns the RW tag of a policy that is for a media of a participant in a conference, if the asking participant has the right to do that.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - a name of a participant
mediaName - the name of the media type rw attribute is placed
policyName - a name of a policy
Returns:
a string containing the rw tag
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be edited or the administrator has not been found
MediaPolicyNotFoundException - if the media type is not available.

PolicyNotFoundException - if the policy is not present
java.lang.Exception

---

setRwPartPolicy
public void setRwPartPolicy(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String policyName,
java.lang.String rw)
throws java.lang.Exception

Sets the RW tag of a policy of a participant in a conference, if the asking participant has got the right to do that.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - a name of a participant
policyName - a name of a policy
rw - a string containing the new content of the rw tag y or n
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information
OutOfRangeException - if the use argument is not y or n
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be edited or the administrator has not been found
PolicyNotFoundException - if the policy is not present
java.lang.Exception

---

setRwMediaPolicy
public void setRwMediaPolicy(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName,
java.lang.String policyName,
java.lang.String rw)
throws java.lang.Exception

Sets the RW tag of a media policy of a participant in a conference, if the asking participant has the right to do that.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - a name of a participant
mediaName - the name of the media type the policy is stored in
policyName - a name of a policy
rw - a string containing the new content of the rw tag y or n
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information
OutOfRangeException - if the use argument is not y or n
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be edited or the administrator has not been found
MediaPolicyNotFoundException - if the media of the named type does not exist
PolicyNotFoundException - if the policy is not present
java.lang.Exception

---

setConfState
public void setConfState(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String stateName,
java.lang.String value)
throws java.lang.Exception

Sets the state of a conference, if the administrator has got the oblige rights. The conference server method confStateChanged is executed to inform the conference server.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
stateName - a name of a participant
value - a string containing the new content of the state
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be edited or the administrator has not been found
PolicyNotFoundException - if the policy corresponding to the state is not present
StateNotFoundException - if the state is not present
java.lang.Exception

---

activateParticipant
public void activateParticipant(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName)
throws java.lang.Exception

Activates a participant by inserting it into the conference states. The conference server method confStateChanged is executed so the conference server is notified.

Parameters:
focusName - A name to identify the administrator
password - A string to identify the administrator
confName - A name of a conference
partName - The name of the participant, that should be activated
Throws:
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be added or the administrator has not been found
java.lang.Exception

---

setPartState
public void setPartState(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String stateName,
java.lang.String value)
throws java.lang.Exception

Sets the state of a participant if the administrator has got the rights. The conference server method partStateChanged is executed to inform the conference server about the changes.

Parameters:
focusName - A name to identify the administrator
password - A string to identify the administrator
confName - A name of a conference
partName - A name of a participant
stateName - A name of a participant
value - A string containing the new content of the state
Throws:
conf.confController.PasswordException - If the password is wrong
conf.confController.NoRightException - If the administrator has not got
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be added or the administrator has not been found

PolicyNotFoundException - If the corresponding policy is not present.
StateNotFoundException - if the state is not present
java.lang.Exception

---

setPartMediaState
public void setPartMediaState(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName,
java.lang.String stateName,
java.lang.String value)
throws java.lang.Exception

Sets the media state of a participant if the administrator has got the rights. The conference server method confMediaStateChanged is called to inform the conference server.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - a name of a participant
mediaName - a name pointing to the type of media the state is saved within
stateName - a name of a participant
value - a string containing the new content of the state
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be added or the administrator has not been found
MediaStateNotFoundException - if the media of type mediaType is not present
PolicyNotFoundException - if the corresponding policy is not present
StateNotFoundException - if the state is not present
java.lang.Exception

---

setConfMediaState
public void setConfMediaState(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String mediaName,
java.lang.String stateName,
java.lang.String value)
throws java.lang.Exception

Sets the state of a conference if the administrator has got the rights. The conference server method confMediaStateChanged called to inform the conference server.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
mediaName - it contains the name of the media type the state is situated
stateName - a name of a participant
value - a string containing the new content of the state
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to change the information
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the participant that should be added or the administrator has not been found
PolicyNotFoundException - if the corresponding policy is not present
MediaStateNotFoundException - if the media of type mediaType is not present
StateNotFoundException - if the state is not present
java.lang.Exception

---

openConference
public void openConference(java.lang.String rootPassword,
java.lang.String confTemplateName,
java.lang.String confName)
throws java.lang.Exception

Creates a new conference out of a template and saves it as a conference with confName. The conference server method conferenceOpened is called to inform the conference server.

Parameters:
rootPassword - the root password
confTemplateName - the name of the template the conference is built after
confName - the sip name of the conference
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.Conference

AlreadyExistException - if a conference with the same is already stored in the database
conf.confController.element.ConferenceNotFoundException - if the conference template you have given does not exist
java.lang.Exception

---

closeConference
public void closeConference(java.lang.String rootPassword,

java.lang.String confName)
throws java.lang.Exception

Removes the conference entrance in the database. The conference server method conferenceClosed is executed to inform the conference server.

Parameters:
rootPassword - the root password
confName - the sip name of the conference.
Throws:
conf.confController.element.ConferenceNotFoundException - if the conference that should be deleted is not present
conf.confController.PasswordException - if the password is wrong.
java.lang.Exception

---

addParticipant
public void addParticipant(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String partName,
java.lang.String partPassword,
java.lang.String partTemplateName)
throws java.lang.Exception

This function adds a participant to the conference if either the rootPassword is given or the administrator has got the right to invite people.

Parameters:
focusName - a name to identify the administrator
password - a string to identify the administrator
confName - a name of a conference
partName - the name the created participant will have
partTemplateName - the name of the template the participant is a copy of
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.NoRightException - if the administrator has not got the right to add a participant
conf.confController.element.ConferenceNotFoundException - if the conference is not present
conf.confController.element.PartNotFoundException - if the administrator is not found or the participant template is not present
conf.confController.ParticipantAlreadyPresentException - if a particiapnt with partName is already registered in the conference
java.lang.Exception

---

removeParticipant
public void removeParticipant(java.lang.String rootPassword,

java.lang.String confName,
java.lang.String partName)
throws java.lang.Exception

Removes a participant entrance in the database. Regard! If the participant is active it is not de-activated.

Parameters:
rootPassword - the root password
confName - The sip name of the conference where the participant, that should be removed is situated.
partName - the name of the participant that
Throws:
conf.confController.PasswordException - if the password is wrong
conf.confController.element.ConferenceNotFoundException - if the conference is not present
java.lang.Exception

---

openSideBar
public void openSideBar(java.lang.String focusName,
java.lang.String password,
java.lang.String confName,
java.lang.String sideBarName,
java.lang.String confTemplateName)
throws java.lang.Exception

Creates a new sidebar, if the focus has got the right. It inserts the focusName as first user with all possible rights. The method sideBarOpened from the conference server is called to notify it.

Parameters:
focusName - the user that likes to create a new side bar
password - the password to identify the administrator
confName - the name of the conference the side bar belongs to.
sideBarName - the name the side bar should have.
confTemplateName - the name of the template the sidebar will be a copy of.
Throws:
conf.confController.PasswordException - if the password is wrong.
conf.confController.NoRightException - if the openSideBar right is not turned on for the administrator.
conf.confController.element.PartNotFoundException - if the administrator is not found
conf.confController.element.ConferenceNotFoundException - if the conference template is not found.
conf.confController.ConferenceAlreadyExistException - if a conference or side bar with that name already exist. Note that it is not possible to have a conference and a sideBar with the same name simultaneously.
java.lang.Exception

---

closeSideBar
public void closeSideBar(java.lang.String focusName,
java.lang.String password,
java.lang.String sideBarName)
throws java.lang.Exception

This function removes a sidebar either if the rootPassword is given or the focus has got the right to open a side bar, within the sidebar itself. The method sideBarClosed of the conference server is executed to inform the conference server.

Parameters:
focusName - the participant that likes to close the sideBar
password - the password of the participant, needed to identify him.
sideBarName - the name of the sideBar that should be removed
Throws:
PartNotFoundExeption - if the participant that likes to close the sideBar has not been found.
conf.confController.PasswordException - if the password is wrong.
conf.confController.NoRightException - if the openSideBar right is not turned on for the administrator.
conf.confController.element.ConferenceNotFoundException - if the sidebar is not present
java.lang.Exception

Conference Server Interface

confStateChanged
public void confStateChanged(java.lang.String confName,
java.lang.String name,
java.lang.String value,
java.lang.String typeOfAction)
throws java.io.IOExceptionThis function receives the notification that a conference state has change.

Parameters:
confName - a String containing the conference name
name - a String containing the name of the state
value - a String containing the new value of the state, if it has been deleted it is empty
typeOfAction - a String indicating the type of action. It contains either a, c, or r. a for added, c for changed and r for removed.
java.io.IOException

---

confPartStateChanged
public void confPartStateChanged(java.lang.String confName,
java.lang.String partName,
java.lang.String name,
java.lang.String value,
java.lang.String typeOfAction)

throws java.io.IOExceptionThis function receives the notification that a participant member state has change

Parameters:
confName - a String containing the conference name
partName - a String containing the participant, where the state has changed
name - a String containing the name of the state
value - a String containing the new value of the state, if it has been deleted it is empty
typeOfAction - a String indicating the type of action. It contains either a, c, or r. a for added, c for changed and r for removed.
java.io.IOException

---

confPartMediaStateChanged
public void confPartMediaStateChanged(java.lang.String confName,
java.lang.String partName,
java.lang.String mediaName,
java.lang.String name,
java.lang.String value,
java.lang.String typeOfAction)
throws java.io.IOExceptionThis function receives the notification that a participant media state has change

Parameters:
confName - a String containing the conference name
partName - a String containing the participant, where the state has changed
mediaName - a String containing the type of the media, the state is situated
name - a String containing the name of the state
value - a String containing the new value of the state, if it has been deleted it is empty
typeOfAction - a String indicating the type of action. It contains either a, c, or r. a for added, c for changed and r for removed.
java.io.IOException

---

confMediaStateChanged
public void confMediaStateChanged(java.lang.String confName,
java.lang.String mediaName,
java.lang.String name,
java.lang.String value,
java.lang.String typeOfAction)
throws java.io.IOExceptionThis function receives the notification that a conference media state has change

Parameters:
confName - a String containing the conference name
mediaName - a String containing the type the state has been changed within
name - a String containing the name of the state
value - a String containing the new value of the state, if it has been deleted it is empty

typeOfAction - a String indicating the type of action. It contains either a, c, or r. a for added, c for changed and r for removed.
java.io.IOException

---

conferenceOpened
public void conferenceOpened(java.lang.String confTemplateName,
java.lang.String confName)
throws java.io.IOExceptionThis function receives the notification that conference has opened.

Parameters:
confTemplateName - he name of the template the conference is built after
confName - he SIP-URI of the conference
java.io.IOException

---

conferenceClosed
public void conferenceClosed(java.lang.String confName)
throws java.io.IOExceptionThis function receives the notification that a conference has closed.

Parameters:
confName - the SIP-URI of the conference
java.io.IOException

---

sideBarOpened
public void sideBarOpened(java.lang.String confName,
java.lang.String sideBarName,
java.lang.String confTemplateName)
throws java.io.IOExceptionThis function receives the notification that a side bar has opened.

Parameters:
confName - the name of the conference the side bar belongs to.
sideBarName - the SIP-URI of the side bar
confTemplateName - the name of the template the initial sidebar will be a copy of
java.io.IOException

---

sideBarClosed
public void sideBarClosed(java.lang.String sideBarName)
throws java.io.IOExceptionThis function receives the notification that a side bar has closed.

Parameters:
sideBarName - the SIP-URI of the side bar
java.io.IOException

Appendix C - Installation Instructions

The policy server uses the Java Xerces implementation to access the XML database file and a SOAP implementation for the communication between the policy server and the rest of the components presented in Figure 2, namely the conference server and the participant agents.

Installation of the external components

Installation of the Xerces Java 2 package

URL: http://xml.apache.org/dist/xerces-j/
CD: /cdrom/xerces-j/

• Copy the **Xerces-J-bin.1.4.4.zip** into the xerces root directory
You can extract the zip files using the Java jar command.

```
jar xf Xerces-J-src.1.4.4
```

• Add the local path of the extracted jar files to your classpath environment variable

Installation of SOAP v2.3 package

URL: http://xml.apache.org/dist/soap/
CD: /cdrom/soap/

• Unpack **soap-bin-2.3.tar.gz** or **soap-bin-2.3.zip** into a desired directory
You can use the following command on unix like systems:

```
gzip soap-bin-2.3.tar.gz
tar -xvf soap-bin-2.3.tar.gz
```

mail.jar URL: http://java.sun.com/products/javamail/
activation.jar URL: http://java.sun.com/products/beans/glasgow/jaf.html
mail.jar CD: /cdrom/soap/javamail/
activation.jar CD: /cdrom/soap/jaf-1.0.2/

• Copy **mail.jar** and **activation.jar** into a desired directory
• Assuming that you extracted the **soap-bin.2.3.tar.gz** into the absolute path *foo*, add to your classpath environment variable the paths ***foo/soap-2_3/lib/soap.jar, mail.jar*** and ***activation.jar***

Installation of package tomcat v3.2

URL: http://jakarta.apache.org
CD: /cdrom/tomcat/

• Copy **jakarta-tomcat-3.2.4.zip** for Windows or **jakarta-tomcat-3.2.4.tar.gz** for Unix systems into a desired directory and unpack it.
• Assuming that you unpacked the compressed file into directory *foo*:

- If you use windows, then line 69 of foo/jakarta-tomcat-3.2.4/bin/tomcat.bat should look like this

```
set CLASSPATH=path-to-xerces\xercesImpl.jar;%CLASSPATH%;%
```

- If use a Unix-like system, then add the following line to **foo/jakarta-tomcat-3.2.4/bin/tomcat.sh** after line 113 (or there about, as long as it is before the export line):

```
CLASSPATH=path-to-xerces/xercesImpl.jar:{CLASSPATH}
```

• Copy the file soap.war located in **foo/soap-2_3/webapps/soap.war** relatively to the SOAP root directory into the directory **foo/jakarta-tomcat-3.2.4/webapps** relatively to the root tomcat directory.
• You should be now be able to list/deploy/undeploy services by pointing your browser to `http://hostname:port/soap`

Installation of the Policy Server

• Unpack **confController-src.1.0.tar.gz** - or **confController-src.1.0.zip** if you are installing the Server in a Windows environment - into a desired directory.
• Change into the directory `confController` relatively to the installation directory.
• Build the code by running the `make` command.
• Copy the files that implement the simple web interface from the directory `confController/servletContainer` into the directory of your web browser, where Java Servlets are expected to be found by the browser.
• Add the directory `confController` to your classpath

Starting the “Policy Server” Service

• Start tomcat by executing the start up script (startup.sh or startup.bat for Windows or Unix respectively), ensuring that the classpath is set as recommended above.

Deploying the Services

• Make sure that the current working directory is confController. Subsequently give the following commands on the command line:

```
%>java org.apache.soap.server.ServiceManagerClient \
http://hostname:port/soap/servlet/rpcrouter deploy \
DeploymentDescriptor.xml

%>java org.apache.soap.server.ServiceManagerClient \
http://hostname:port/soap/servlet/rpcrouter deploy \
DeploymentDescriptorAI.xml
```

• As a last step you have to deploy the services implementing the functions on the Conference Server.

Configuration of application properties

• The paths to the database files in the file `confController.conf` in the directory `confController` have to be set as follows:

```
database=path-of-confController/confController/db/policy.xml
databaseConfTemplateFile=path-of-confController/confController/ \
db/conference.template.xml
databasePartTemplateFile=path-of-confController/confController/ \
db/participant.template.xml
serviceName=urn:conferenceServerServiceName
```

• The configuration file `admin.conf` of the simple web interface in the directory where the corresponding servlets are hosted must be configure as follows:

```
conferenceServerHost=hostname:port
serviceNameAdministrationInterface=urn:confControllerAdmin
serviceNameProvideConferenceStatus=urn:confControllerStatus
```