Using Web Services for Application Development in Internet Telephony: A Case Study on Conferencing in SIP Networks

May El Barachi

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

March 2004

Your file Votre référence
ISBN: 0-612-91023-7
Our file Notre référence
ISBN: 0-612-91023-7

ABSTRACT

Using Web Services for Application Development in Internet Telephony:
A Case Study on Conferencing in SIP Networks

May El Barachi

Applications offered to end-users as value-added services are a critical element for the success and the survival of Internet telephony service providers. Some service creation frameworks have been proposed by IETF while alternative frameworks have been proposed by other forums such as Parlay and JAIN. All these frameworks have several drawbacks which may be solved by the use of web services. Web services are self-describing, modular business applications that expose the business logic as services over a network, generally, the Web. This thesis presents a case study on the use of web services to develop conferencing applications in SIP networks. The case study involves the definition of comprehensive web service interfaces that make conferencing capabilities available to developers. It also involves the implementation a sub-set of the interfaces as a gateway using SIP servlets, the development of two applications using the interfaces, and performance evaluation. The applications we have developed consist of dial in and dial out audio conferences. The interfaces proposed offer a level of abstraction that is higher than what the standard frameworks provide. In addition, their use does not require a background in telephony or telecommunications. This could allow developers who are new to the telecom field, to easily integrate conferencing capabilities to their applications. Furthermore, a considerable reduction of the application footprint is achieved. This reduction saves time and effort for application developers. These benefits make web services an interesting approach, despite the overhead introduced.

ACKNOWLEDGEMENTS

I would like to express my gratitude for my supervisors Dr. Rachida Dssouli and Dr. Roch Glitho for their patience, guidance, and great support. Dr. Dssouli, you are an example to all of us, and your dedication and kindness are truly unique. Dr. Glitho, your motivation and guidance are what pushed me to go through this research. Despite your busy schedule, you are always there for your students. Thank you for teaching me so much.

I would also like to thank all my team mates who worked with me in the TSE research lab. All of you have been such a great company and of great help when I needed you. I wish you all the best of luck. Many thanks to the SINTEL team members who provided me with invaluable resources and support.

I would also like to acknowledge funding from the Concordia Research Chair in Telecommunications Software Engineering.

Words are not enough to express my gratitude to my parents and my brother who always encouraged me to push my limits and have confidence in myself and my work. To my husband Amir and my daughter Yasmine, you are the light of my life. I am blessed to have both of you.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ACRONYMS AND ABBREVIATIONS

3GPP: Third Generation Partnership Project

API: Application Programming Interface

CGI: Common Gateway Interface

CPL: Call Processing Language

CPXe: Common Picture eXchange environment

ECMA: European Computer Manufacturer's Association

ETSI: European Telecommunications Standards Institute

GIS: Geographic Information Systems

H.323: Internet video conferencing standards from ITU-T

HTTP: Hyper Text Transfer Protocol

IETF: Internet Engineering Task Force

ISDN: Integrated Services Digital Network

ITU-T: International Telecommunication Union - Telecommunication

JAIN: Java APIs for Integrated Networks

JCC/JCAT: Java Call Control/Java Coordination and Translation

LAN: Local Area Network

MCU: Multipoint Control Unit

Megaco: MEdia GAteway COntrol

MG: Media Gateway

MGC: Media Gateway Controller

MMS: Multimedia Messaging Service

OASIS: Organization for the Advancement of Structured Information Standards

OMA: Open Mobile Alliance

OSA: Open Service Architecture

OWSER: OMA Web Services EnableR

PSTN: Public Switched Telephony Network

RSVP: Resource reSerVation Protocol

RTP: Real-time Transport Protocol

SAP: Session Announcement Protocol

SDP: Session Description Protocol

SIP: Session Initiation Protocol

SMS: Simple Messaging Service

SOA: Service-Oriented Architecture

SOAP: Simple Object Access Protocol

TCP: Transport Control Protocol

UA: User Agent

UAC: User Agent Client

UAS: User Agent Server

UDDI: Universal Description, Discovery, and Integration standard

UDP: User Datagram Protocol

URI: Universal Resource Identifier

URL: Uniform Resource Locator

W3C: World Wide Web Consortium

WSDL: Web Services Definition Language

WS-I: Web Services Interoperability organization

XML: Extensible Markup Language

Chapter 1

Introduction

This chapter first gives an introduction to the research area. It then presents the problem that will be tackled and the solution we propose. The aim of this chapter is to give the reader a flavor of the work to be presented in the rest of the thesis.

1.1 Overview

Beside cost savings, there are other driving forces behind Internet telephony. One of the important motivations is the integration of voice and data applications, which can enable a wealth of new value-added service possibilities. These services are a critical element for the success and the survival of Internet telephony service providers. They are usually divided in two groups: telephony services and non-telephony services. Telephony services interact with call control which enables the initiation, the modification, and the tear down of calls. Examples of telephony services include call forwarding, call screening and conferencing. Non-telephony services, such as customized stock quotes and web surfing from cellular phones, do not interact with call control. Our work here is mainly concerned with telephony services. These services are offered to end-users in the form of applications.

Two main sets of standards exist for Internet telephony: H.323 from the ITU-T [16] and the Session Initiation Protocol (SIP) [32] from the IETF. References [24, 38] provide overviews about these standards. H.323 embraces a traditional approach for signaling, based on the integrated services digital network (ISDN) architecture. SIP on the other hand flavors a lightweight approach, drawing heavily on existing Internet tools. It has been selected by the third-party generation partnership (3GPP) [42] as the sole signaling system

for third generation networks. For that reason, we chose SIP as the signaling environment for our case study.

Several service creation frameworks exist for the development of value-added services. The IETF has proposed two frameworks: The Call Processing Language (CPL) and the SIP Common Gateway Interface (SIP CGI). Alternative frameworks have been proposed by the W3C, the JAIN, and the Parlay forums. Among these frameworks, we find the Parlay API, the SIP Servlet API and the JAIN Java Call Control (JCC)/ Java Coordination and Translation (JCAT). Reference [10] provides an overview of the standard service creation frameworks for Internet telephony and pinpoints their weaknesses.

The web service technology is now emerging and being applied to several industries. It adopts a service-oriented architecture (SOA) where all of the applications are encapsulated as services and made available for invocation over a network. Expected benefits of this technology include loose coupling between different components, reduction of the complexity by encapsulation, and ease of applications development and integration. Standardization work is carried by several standards bodies such as the World Wide Web Consortium (W3C) [49], the Organization for the Advancement of Structured Information Standards (OASIS) [28], and the Web Services Interoperability organization (WS-I) [48].

An industry that is in the process of adopting Web services is telecommunications. The standards being developed include Parlay X [30], ECMA's specifications [6], and the Open Mobile Alliance (OMA) specifications [27].

1.2 Purpose of the Thesis

Applications offered to end-users as value-added services are a critical element for the success and the survival of Internet telephony service providers. Several service creation frameworks exist, however they have many drawbacks. This thesis presents a case study on the use of Web services for the development of conferencing applications in Internet Telephony, or more precisely in SIP networks. By this approach, we try to overcome some of the drawbacks exhibited by the standard frameworks. For instance, a common characteristic between these frameworks is that they require knowledge that non-experts may not have (e.g., scripts, programming languages, APIs). Exposing the needed capabilities, such as call control, as web services may solve this problem.

In this case study, we have defined comprehensive web service interfaces that make conferencing capabilities available to developers. In addition, a sub-set of these interfaces was implemented as a SIP servlet based gateway; the SIP Servlet API being more commonly used in a SIP environment than other frameworks. The sub-set implemented is related to pre-arranged conferencing functionalities. Two applications were built using the implemented interfaces and performance was evaluated. The applications built consist of dial-in and dial-out audio conferencing applications.

The interfaces proposed offer a level of abstraction that is higher than what is provided by the standard frameworks. This level of abstraction could allow developers who are new to the telecom field, to easily incorporate conferencing capabilities to their applications. Another important merit is the considerable reduction of the application footprint. This reduction saves time and effort for application developers. These benefits make web services an interesting approach, despite the overhead generated.

1.3 Organization of the Thesis

In chapter 2, we start by providing background information about the different conferencing models, the signaling environment used (SIP), and the web service technology that may ease the development of conferencing applications.

Chapter 3 presents the state of the art in service creation frameworks that can be used for the development of conferencing applications in SIP networks. The standard service creation frameworks are first discussed. We then move to the emerging service creations frameworks, based on the web service approach. The SIP servlet API framework is emphasized, since it is the framework used for the building of the web services gateway.

Chapter 4 is dedicated to the description of the case study components. Among the components presented, we focus on the description of the new interfaces and the mapping of a sub-set (pre-arranged conferencing functions) onto SIP.

In chapter 5, the components software architecture and the prototypes built are presented. The performance measurements are then reported and analyzed.

The thesis concludes with chapter 6, which contains a recapitulation of the major work of the thesis, the lessons learned, and suggestions for future work.

Chapter 2

Review of Conferencing Models, SIP, and Web Services

Conferencing is a critical application area for Internet Telephony. It includes audio/video conferencing, but also multiparty gaming, and distance learning. In this chapter, we successively introduce it, present the signaling environment used in this case study (SIP), and discuss Web services a technology that may ease the development of related applications. The aim of this chapter is to define the general context of the work presented in this thesis.

2.1 Conferencing models

The two main aspects of any conferencing architecture are signaling and media mixing. Several topologies can be used to perform these two tasks. Other capabilities such as floor control can also be used. The latter allows users of networked multimedia applications to utilize and share resources (e.g., media channels) without access conflict [4]. Another criterion for distinguishing between the different models is the way the conference starts. While pre-arranged conferences have a pre-determined start time and a conference identifier, ad-hoc conferences are more spontaneous and begin as soon as the first two users start communicating. In this Section, we give an overview about the commonly used conferencing models. These models are categorized based on their media distribution topologies. Figure 2.1 illustrates these models and table 2.1 summarizes their properties. In addition, references [37, 34, 39] provide more details about these models, in the SIP context.

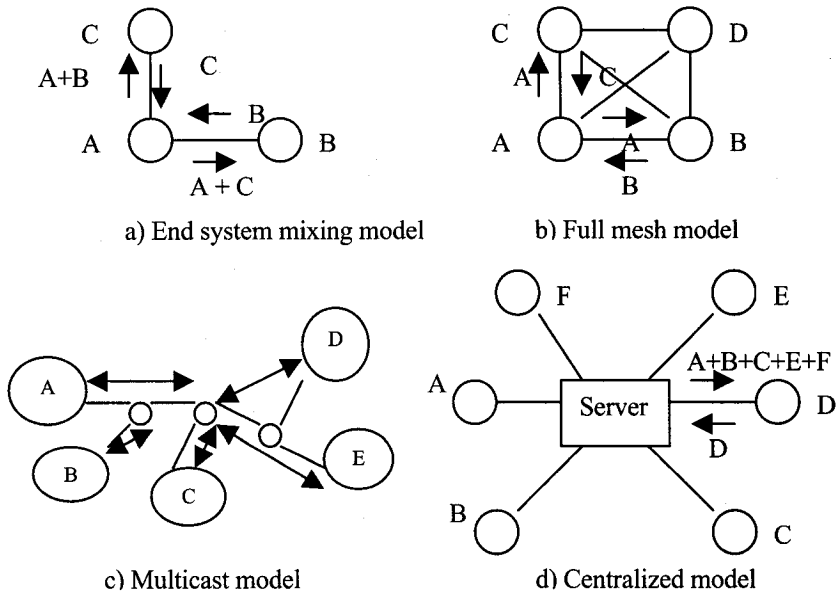


Figure 2.1: Conferencing models topologies.

2.1.1 The End System Mixing Model

In this model, signaling and media mixing are handled by one of the participants. For example, A calls B then later calls C, using a different session. There is no direct call setup between B and C. A receives media streams from B and C, and mixes them. It then sends the mixed streams to the appropriate users. B and C do not need to be aware of the service provided by A, but can in turn mix other participants. In fact, any user who has media mixing capabilities can invite other users to the conference.

This model is built around ad-hoc conferences. Its main drawback is that the conference ends when the mixing participant leaves the call. Another problem is scalability. Due to the limitation of the mixing workstations capabilities, this model is more likely to scale only for small conferences.

2.1.2 The Full Mesh Model

In a full mesh, every participant builds a signaling leg with every other participant and sends an individual copy of the media stream to the others. The participants' end systems sum the incoming streams. This mechanism only scales to very small groups. In this model, each pair of participants must share a common codec.

2.1.3 The Multicast Model

In this case, a multicast address is allocated to the conference. Each participant joins that multicast group and sends his media to it. Signaling is not sent to the group, it is only used to inform participants to which multicast group to join. Like the full mesh case, participants receive packets on the same address from all other participants, and need to sum streams. On the other hand, each end-system needs to generate only one copy of its own stream, to send to the multicast address.

Multicast conferences are usually pre-arranged. However, the multicast model can also be used for ad-hoc conferences, as long as a mechanism exists to dynamically obtain a multicast address. This model scales very well to large-scale conferences. Unfortunately, it has not been widely deployed across backbones. As a result, wide area conferences are not really viable using multicast.

2.1.4 The Centralized Model

In this model, a server or bridge receives streams from all the participants, mixes them, and redistributes the appropriate stream to each user. The participants' end systems do not require any special processing. All the new logic resides in the conferencing server. Participants either call the server (dial-in model) or are called by the server (dial-out

model). Several applications could be built using these models. The dial-out model is suited for conferences directed by a chair and multiparty games initiated by game servers, while the dial-in model is more suited for online public debates and chat applications.

The server based model offers some advantages. First, the clients do not need to be modified and do not need to perform media mixing. In addition, heterogeneous media types can be supported, since the server performs decoding. The server can also enforce floor control policies. Finally, this model can be used for both pre-arranged and ad-hoc conferences.

Models / Properties	End Mixing	Full Mesh	Multicast	Centralized
Signaling topology	Tree	Full mesh	Pairs	Star
Media topology	Tree	Full mesh	m-cast tree	Star
Scaling	Small	Small	Large	Medium
Heterogeneous endpoints	Yes (Partially)	Yes	No	Yes
Starting mode	Ad-hoc	Pre-arranged	Pre-arranged/ Ad-hoc	Pre-arranged/ Ad-hoc

Table 2.1: Conferencing models properties.

2.1.5 Discussion

Centralized conferences are easier to handle for end systems and simplify keeping track of the conference participants. On the other hand, multicast conferences have an excellent scalability for large conferences and allow a "loose" model of conference membership. However, as long as multicast is not widely deployed, centralized conferences will remain the only viable model for mid-size conferences of tens to hundreds of participants. It is therefore the most appropriate model for Internet telephony applications. We based our case study on centralized conferencing for that reason.

2.2 The Session Initiation Protocol

The Session Initiation Protocol (SIP) is an application-layer signaling/control protocol for creating, modifying and terminating multimedia sessions with one or more participants. SIP extensions are used for event notification and management of other types of sessions, such as distributed games. Core SIP was published as a proposed standard (RFC 2543) [13] in March 1999, and June 2002 (RFC 3261) [32]. Other standards exist for SIP extensions such as RFC 3265 [31] and 3515 [41] (for the event framework), and RFC 2976 [5] (for the INFO method).

SIP is designed as part of the overall Internet Engineering Task Force (IETF) multimedia architecture. Examples of other IETF protocols with which SIP is used in conjunction include: quality of service related protocols (e.g., RSVP); media transport protocols (e.g., RTP); and other protocols (e.g., SDP and SAP). References [38, 33, 36, 20, 40] constitute a list of some of the articles and books published on SIP.

2.2.1 SIP Main Characteristics

In this section, we present some of the important features of the session initiation protocol. SIP is based on HTTP, reusing many of its header fields, encoding rules, error codes, and authentication mechanisms. Like HTTP, SIP is a request/response protocol in which each operation consists of a request going from the client to the server followed by a response returned from the server to the client. SIP is also a text-based protocol. This textual nature promotes easy development and debugging of SIP based applications, in addition to making SIP quite flexible and extensible.

SIP messages can be transported on any transport protocol (e.g., UDP and TCP). SIP applications can range from traditional call waiting and 800 numbers services to multimedia conferencing, presence, and distributed games. Finally, SIP supports both unicast and multicast sessions, and can also initiate multiparty calls using a Multipoint Control Unit (MCU) or a fully meshed connection.

Five facets of multimedia communications are supported by SIP: user location, user capabilities, user availability, call setup, and call handling. User location consists of the determination of the end-system to be used for the communication. In SIP, user capabilities are assessed by determining the media type and parameters to be used. User availability support consists of determining the called party's willingness to engage in the communication. Call setup is accomplished by the establishment of the call parameters of both called and calling party, and call handling includes transfer and termination of calls.

2.2.2 SIP Architecture

Two main components constitute the core of the SIP architecture: User agents (UAs) and network servers. A user agent is an end-system which takes order from a user and acts on his behalf, to setup and tear down sessions with other user agents. Usually the UA consists of two parts, a client (UAC) and a server (UAS). These components enable the user to both place a call and be called. The UAC is used to initiate requests while the UAS receives requests and returns responses on behalf of the user.

On the other hand, network servers are applications that accept SIP requests and respond to them. They provide services to user agents. There are three kinds of network servers: proxy servers; redirect servers; and registrars. Proxy servers act as routers, forwarding SIP

requests and responses. They usually use backend location servers to map a request URI to a new destination. Typically, SIP messages originating at a user agent traverse one or more SIP proxy servers, then reach one or more SIP user agents. However, user agents can also communicate directly between each others. A redirect server receives requests then returns the address of another server or UA where the client might be found. Finally, registrars keep track of users registered with them. Figure 2.2 shows these elements and their interactions.

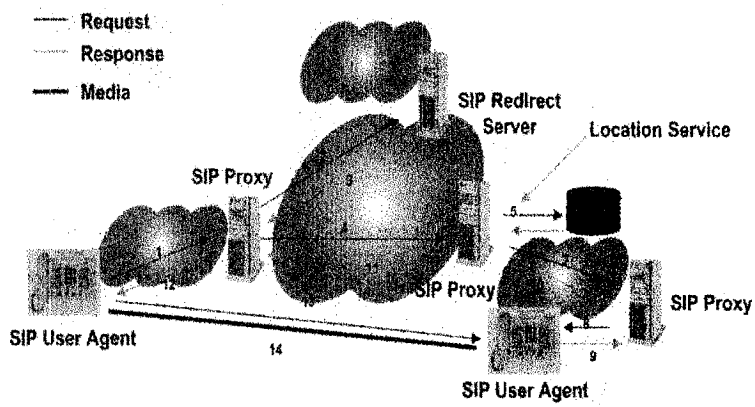


Figure 2.2: SIP basic architecture.

2.2.3 SIP Messages

A SIP message is either a request from a client to a server or a response from a server to a client. Figure 2.3 depicts the general structure of SIP messages.

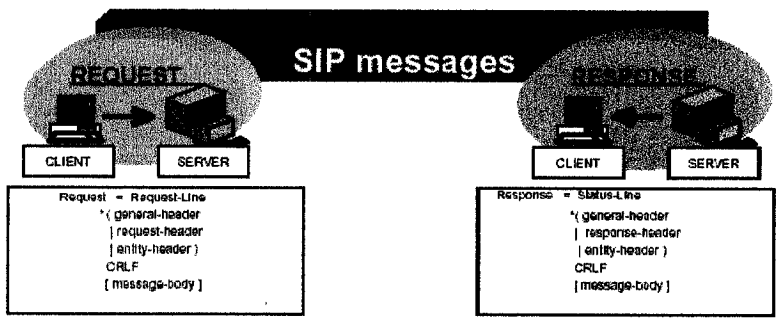


Figure 2.3: SIP messages structure.

Request messages start with a request line, which includes the request method, the user or server to which the request is being addressed (request URI), and the version of SIP used. Response messages on the other hand start with a status line. This line includes a three digit status code indicating the response type, and a reason phrase giving a short text description of the status code. Besides the first line, both types of messages include several headers and sometimes a message body.

2.2.3.1 Header Fields

Four types of headers can be included in SIP messages: General headers indicate general information that can be included in both requests and responses. The "To" and the "From" headers are examples of general headers. On the other hand, entity headers provide information about the message body, such as the content length and the content type. Request headers allow the client to pass additional information about himself and about the request, to the server. The contact header, indicating an alternative address where the client can be reached, is an example of request headers. Finally, response headers allow the server to pass additional information about the response, which cannot be placed in the status line. A list of the headers used in this work and their description is presented in table 2.2. References [32] can be consulted for more details about the basic and the extension headers available.

Header fields	Type	Description
To	General header	Specifies the desired recipient of the request
From	General header	Specifies the initiator of the request
Via	General header	Indicates the path used by the request. The same path is traversed by the response, in reverse direction
Call-ID	General header	Provides a unique ID identifying a certain session
Cseq	General header	Indicates the request method and a sequence number identifying the different transactions

Contact	General/Request header	Provides an alternative address where the user can be reached
Subject	General/Request header	Indicates the message subject
Content-Length	Entity header	Indicates the size of the message body, in number of octets
Content-Type	Entity header	Indicates the media type of the message body

Table 2.2: SIP Header fields used.

2.2.3.2 Message Body

All SIP messages may include a message body, except the CANCEL request message. Usually, the message body includes a session description based on the session description protocol (SDP) [14]. This description conveys the information necessary to allow a party to joins a multimedia session (session and media related information). When a message contains a body, the body length and type must be indicated in the Content-Length and Content-Type headers respectively. If the message is encoded, this should be indicated in the Content-Encoding header.

2.2.3.3 Request Methods

In Core SIP, six possible methods can be used is a request message: INVITE, ACK, CANCEL, BYE, REGISTER, and OPTIONS. Note that the first four methods are used for setting up sessions while the last two methods are used for registrations and capability queries. A description of these methods is presented as follows:

- INVITE: This method indicates that a user or a service is being invited to participate in a session. A media session is considered established when INVITE, 200 OK, and ACK messages have been exchanged between the UAC and the UAS.
- ACK: Confirms that the client has received a final response to an INVITE request. This method is only used following an INVITE method.
- CANCEL: This method is used to cancel a pending request. It does not affect a completed request, a request that has been processed, or a response that has been sent back.

- BYE: The UAC uses this method to indicate to the server that it wishes to release the call. This request can be issued by either the caller or the callee.
- REGISTER: This method is used by a client to bind his address with one or more URIs where he can be reached. This request is sent by the client to a registrar.
- OPTIONS: This method is used to query a server about its capabilities. The response can include headers such as Accept and Allow.

In addition to these methods, other methods have been introduced as extensions to SIP. Examples of these extensions include: the SUBSCRIBE; NOTIFY; and INFO methods. The SUBSCRIBE method is used by a requestor for subscription to certain event(s). The NOTIFY method is used by the provider to send notifications to the requestor when the event(s) occur. Note that SUBSCRIBE can also used for un-subscriptions by setting the expiry header to zero. The INFO method is used for the exchange of additional information, that can't be conveyed by standard messages, between parties.

2.2.3.4 Response Codes

Six types of responses are possible in SIP. Each type is distinguished by a specific digit starting the status code. These types are summarized as follows:

1xx: **Informational** - the request was received and is going to be processed.
2xx: **Success** - the action was successfully received, understood and accepted.
3xx: **Redirection** - further actions need to be taken in order to complete the request.
4xx: **Client Error** - the request contains bad syntax or cannot be fulfilled at this server.
5xx: **Server Error** - the server failed to fulfill an apparently valid request.
6xx: **Global Failure** - the request cannot be fulfilled at any server.

Examples of the available response codes include the 180 response code, which means that a client is sending a ringing or that it has been alerted by the request. Another example is the 200 response code, which indicates that the client accepts the request.

2.2.4 Alternatives to SIP

In this section, we give a brief overview of two of the alternatives to SIP: H.323 and Megaco/H.248.

H.323 is an umbrella standard including signaling standards (e.g., H.225, Q.931, and H. 245) and other standards (e.g., H.324). H.323 functional entities consist of: terminals; gatekeepers; gateways; and MCUs. A terminal is an end-point which is used in two-way real time multimedia communication with another end-point. A gatekeeper provides address translation and controls how a terminal accesses the network. Gateways are used in the communication between H.323 terminals and terminals in the Public Switched Telephony Network (PSTN). An MCU provides centralized conferencing functionality. The H.323 recommendation is available at [16].

Megaco/H.248 is mainly used for the decomposition of gateways between Internet telephony networks and circuit switched telephony networks. Two important concepts in Megaco/H.248 are terminations and contexts. A termination is a source or a sink of media flow. A context is a mixing bridge to which a number of terminations are connected. Terminations can be added to a context or removed from a context. Other operations such as modification of the properties of a termination, moving a termination between two contexts, auditing termination capabilities, notifications and change of services are supported by this protocol. Megaco/H.248 can also be used for telephony signaling. In this case, the IP phone is a Media Gateway (MG) controlled by the Megaco/H.248 protocol. The call control intelligence is located in the Media Gateway Controller (MGC) which acts as a soft switch in this case. Reference [12] provides more details about Megaco/H.248.

2.3 Web Services

The web services architecture has been introduced by the World Wide Web Consortium [45]. Web services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the Web. This section provides an overview about the Web service technology. Reference [35] provides an overview and reference [25] provides tutorial-level information.

2.3.1 Web Services Main Characteristics

The different aspects of a web service are summarized as follows: A web service is an interface that describes a collection of operations, which are network-accessible through standardized Extensible Markup Language (XML) messaging. It is described using a standard XML notation, called its service description, expressed in Web Services Definition Language (WSDL). This description covers all the details necessary to interact with the service, including message formats, transport protocols, and location. In addition, this description can be published to a service registry using the Universal Description, Discovery, and Integration standard (UDDI), where it can be discovered later by a service requestor. Applications use the Simple Object Access Protocol (SOAP) as a communication protocol to interact with web services. SOAP messages are typically carried over HTTP, but can use other transport protocols. The web service interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform, on which it is implemented, and of the programming language in which it is written. Finally, web services fulfill a specific task or a set of tasks. They can be used alone or with other web services. References [47, 46, 26, 44] can be consulted for XML, WSDL, UDDI, and SOAP specifications.

2.3.2 The Web Services Architecture

The web services architecture places into relationship different components and technologies. Figure 2.4 illustrates these components, their operations and interactions.

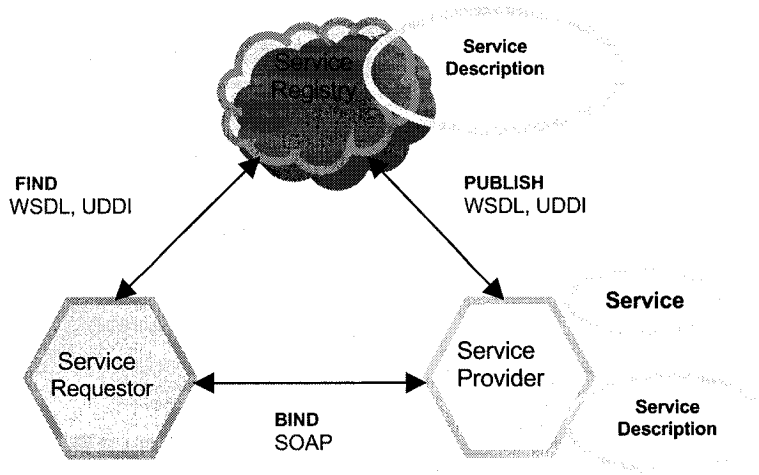


Figure 2.4: Web services roles, operations and objects.

Typically, a service provider hosts a network accessible software module (an implementation of a web service). In addition, the provider defines a service description and publishes it to a service registry, based on the UDDI specification. Once a web service is published, a service requestor may find the service via the UDDI interface. The UDDI registry provides the service requestor with a WSDL service description and a URL pointing to the service itself. The service requestor may then use this information to directly bind to the service and invoke it.

2.3.3 The Web Services Development Lifecycle

Generally, the web services development lifecycle is composed of four phases: building, deploying, running, and managing the services. These phases are described as follows:

- **Web service Building:** This phase consists of the development and testing of the web service implementation, and the definition of the interface description and the implementation description. The implementation can be used to create new web

services, to transform existing applications into web services and to compose new web services from other web services and applications.

- **Web service Deployment:** This phase starts with the deployment of the executable modules into an executable environment (e.g., a web application server). Afterwards, the service interface and the service implementation definition must be published to a service requestor or service registry.
- **Running the Web service:** At this stage, the web service is available for use. It has been fully deployed, and is operational and ready to be network-accessed by the requestor.
- **Web service Management:** This phase deals with the management and the administration of the web service, by treating issues such as security, performance, quality of service, and service composition.

2.3.4 Expected Benefits of the Web Service Technology

In this section, we give an overview of the key benefits expected from the web service technology. These benefits are summarized as follows:

1. *Loose coupling between different components:* In the web services architecture, all components are loosely-coupled, and binding occurs at run time. This implies that a change in the implementation of the services does not require any changes in the applications using them. This loose-coupling is translated in more flexibility, scalability, and extensibility.
2. *Reduction of the complexity by encapsulation:* The web services technology adopts a Service Oriented Architecture (SOA) where all applications are exposed as services. This coarse-grain approach puts the focus on the service behavior, rather than the service implementation, thus reducing the complexity of the services used by the developers.
3. *Easy application development and integration:* The high level of abstraction provided by this technology allows developers to integrate the needed functionality to their applications easily and rapidly. In addition, no knowledge of the underlying architecture of these functionalities is needed.

2.3.5 Examples of Web Services Application Domains

The web service technology is now emerging and being applied to several domains besides telecommunications. Digital imaging and geographic information systems (GIS) are examples of these domains. In the digital imaging industry, the common picture exchange environment (CPXe), created by a large number of imaging companies, is based on the web service technology. Its main goal is to offer consumers a broad range of digital imaging services, while giving providers access to an expanded market. In this framework, services such as printing, resizing, image layering, and shipping, are exposed as web services. Their WSDL descriptions are published in a UDDI registry, to be discovered by service requestors. In addition, catalog services and service locators are provided to facilitate service discovery. After discovering these service descriptions, requestors can place their orders via the provided interfaces. Reference [43] gives an overview of this framework. Similar work is done in the GIS field. The goal of this work is to dynamically assemble applications from multiple GIS web services, for use in a variety of client applications. Examples of services to be exposed include reprojection, overlay, and portrayal services. Reference [1] gives more details about this framework.

Chapter 3

Conferencing in SIP Networks: State of the Art

In this chapter, we present the state of the art in service creation frameworks that can be used for developing conferencing applications in SIP networks. We first present an overview of the standard service creation frameworks for Internet telephony. We then introduce the emerging frameworks proposed by the telecom industry in order to adopt the web service technology. We put a special emphasis on the SIP servlet API, since it is the framework we chose for the building of the web services gateway.

3.1 The Standard Service Creation Frameworks

The standard service creation frameworks can be divided in two categories: signaling protocol neutral frameworks, and signaling protocol specific frameworks. We'll now give an overview of the frameworks belonging to each of these categories.

3.1.1 Signaling Protocol Neutral Frameworks

Three main signaling protocol neutral frameworks exist: The Call Processing Language (CPL), JAIN's JCC/JCAT, and the Parlay/OSA API. As the name implies, these frameworks can be used with any signaling protocol, including H.323 and SIP. The Parlay/OSA API is the most deployed framework in this category.

3.1.1.1 The Call processing language

The Call Processing Language [22] has been proposed by IETF. It targets primarily end-users but also service providers. Users can directly write CPL scripts, defining services, and upload them to a network server. A transport mechanism is needed to upload the scripts

from end-systems (e.g., H.323 terminals and SIP clients) to servers. The use of the REGISTER message, in the case of SIP, has been proposed for this purpose [23]. Scripts should be able to react to signaling events. In the case of a call invitation for instance, the script should be able to reject the call, redirect it, or proxy it. At the network server, the scripts can be verified and the service instantiated instantly.

3.1.1.1.1 The Language Requirements

Since CPL is made to be used by untrusted users, some requirements had to be imposed on the language: These requirements are summarized as follows:

- Verifiability: Upon receipt of a script, the service provider must be able to verify its correctness and that the server can successfully execute it.
- Completion: The service provider must also be able to determine that the service specified in the script will be completely executed in a finite amount of time. This implies the absence of general loops and of calls to external services without timeouts.
- Safety of execution: The service should be executable in a safe manner. For instance, it should not make an excessive use of the server resources (memory, bandwidth...and so on) and should not manipulate sensitive data on the server.
- Standardized representation: Service descriptions must be compatible between different tools and must be producible by both humans and machines.

3.1.1.1.2 The Key Features

CPL represents services in a decision graph. The graph's individual nodes are the primitives of the language. They represent decisions to be made or actions to be taken during the service execution. Each node may have output (s) which leads to further actions or decisions. After a node execution, the script may follow one of the outputs or terminate, depending on the result. Figure 3.1 shows a CPL decision graph for a call forward service.

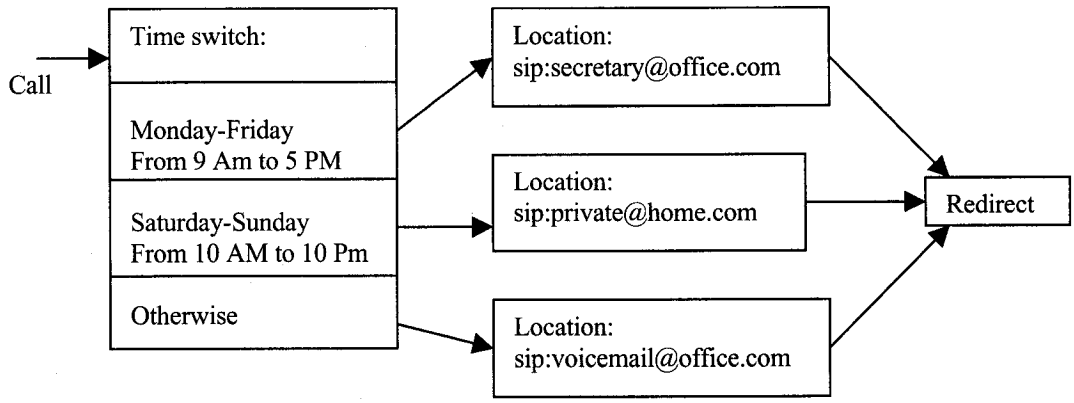


Figure 3.1: Example of a CPL decision graph.

This node-based structure implicitly guaranties most of the mentioned requirements. Since the flow of control moves only downwards, this implies that the service will eventually reach a node and terminate. In the worst case, the resources consumed by the service are proportional to the length of the longest branch of the graph, and are therefore finite. The language contains no loops, no function calls, and no access to external programs. This makes the service executable in a safe manner.

CPL scripts are based on XML because of several important features it offers. XML consists of a hierarchy of tags, each tag having a set of attributes. It is therefore suitable for the representation of structured data and tree structures. XML can be produced and read by both humans and machines. It can also be verified automatically. Finally, XML has no specific key words. This allows the definition of the needed primitives.

3.1.1.1.3 The Language Primitives

There are four types of primitives in CPL: switch nodes, location nodes, signaling action nodes, and non-signaling action nodes. Switch nodes represent decisions a script can make. They match call attributes or call independent parameters against a list of conditions. Each condition corresponds to a certain output of the node.

Location nodes specify the locations needed for call processing. A basic location node specifies the location as a URL and has one output. A location lookup node determines the location by consulting an external source (e.g., a SIP registrar or a database) and has three possible outputs (success, failure, and not found).

Signaling action nodes control the behavior of the underlying signaling protocol. There are three types of signaling action nodes: proxy, redirect, and response nodes. A proxy node makes the server forward the call to the currently specified set of locations. The server then waits for the responses and selects the best response. Possible outputs include success, busy, no-answer, and failure. Redirect and response nodes immediately terminate the execution of the script. A redirect node makes the server redirect the calling party to the currently specified set of locations. A response node makes the server send a failure response or reject the call.

Non-signaling action nodes allow the script to log events and notify the user of them. For instance, a script can send an instant message to a user, warning him about a malfunction which is preventing him from receiving calls.

3.1.1.1.4 Pros and Cons

The main advantage of CPL is that it allows end-users to create their own services. In addition, it provides an environment in which these services can be executed safely. However, it has several drawbacks. First, the absence of loops and function calls makes it tedious to create services with repetitive patterns. In addition, creating services that access external data is not possible. Finally, the range of services that can be created is limited. In fact, due to the limitations of the language, CPL can not be used for the development of

complex applications such as conferencing applications. The language primitives are designed to be simple and have a specific behavior, as described previously. These primitives cannot be used to manage a conference and keep track of multiple signaling objects such as participants, ports and so on. Therefore, the capabilities offered by these primitives limit the range of the possible services to simpler services such as call redirect, call forward, call screening, and time of the day routing.

3.1.1.2 JAIN's JCC/JCAT

The Java Call Control (JCC)/ Java Coordination and Translation (JCAT) framework gives access to call control capabilities. The JCC API provides the core functionality, while the JCAT API extends it by providing a finer granularity. Examples of the services that could be developed using these APIs are: first and third party calls; toll free number translation; voice activated dial; and dial-in conferences. Reference [17] provides an overview of this framework.

The JCC/JCAT call control model is composed of the following objects: a provider; a call; a connection; and an address. The call is an abstraction of the physical call. A provider is an access point used by the service to view a call. The address represents a call party while the connection is a logical link between the call and the address.

Due to its signaling protocol neutrality, the JCC/JCAT framework enables the creation of services that are portable across network technologies. However, the following drawbacks are observed: the framework is not programming language neutral (tendency towards Java); the level of abstraction offered is low and necessitates a background in circuit switched telephony; security and resilience features are not offered. We should mention

that the JCC/JCAT framework has lost momentum to the Parlay framework presented in the next section.

3.1.1.3 The Parlay/OSA API

The Parlay/OSA API was originally defined within the Parlay group and standardized by 3GPP and the European Telecommunications Standards Institute (ETSI) [7]. So far, four releases of Parlay/OSA specifications have been released. Reference [29] can be consulted for the latest release. The main goal of the Parlay/OSA API is to open up telecommunication networks in a secure and resilient way. This is accomplished by making network capabilities available for applications development. In addition, the API enables new business models and is based on open information technology.

3.1.1.3.1 The Parlay Business Model

There are three roles in the Parlay business model: client application; enterprise operator; and framework operator. The client application uses the services (network capabilities). Prior to the service usage, a subscription to the service should be made. The enterprise operator is the entity that subscribes to the services, while the framework operator handles the subscriptions. Figure 3.2 illustrates the Parlay business model.

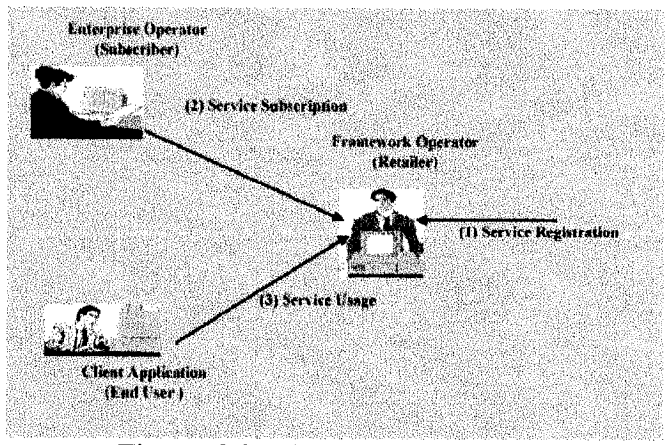


Figure 3.2: The Parlay business model.

3.1.1.3.2 The Parlay APIs

There are two types of APIs in Parlay: service APIs and framework APIs. The service APIs expose the network capabilities. They include interfaces for call control, user interactions, generic messaging, mobility, terminal capabilities, connectivity management, account management, charging, session control, presence and availability. The call control interfaces are the most relevant for Internet telephony applications. The framework APIs make the use of the service APIs secure and resilient. They include interfaces for trust and security management, event notification, service discovery, service registration, integrity management, and service agreement.

A. The Call Control APIs

In order to give an example of the provided interfaces, we'll describe the call control interfaces since they are the most relevant to the scope of this thesis. The call control API is made of four packages: a generic call control package; a multiparty call control package; a multimedia call control package; and a conference call control package.

Three types of objects are manipulated by these packages: a call, a call leg, and an address object. The call and the address objects have the same significance as in the JCC/JCAT framework. The call leg is similar to the connection object used in JCC/JCAT.

The generic call control package supports two-party calls only. It remains in Parlay for historical reasons. The multiparty call control package permits the establishment of calls with any number of users. It also allows operations on call legs. It is the root of the inheritance tree. The multimedia call control package extends the multiparty package by adding multimedia capabilities. The conference call control package adds another extension

by adding support to conferencing capabilities. Figure 3.3 illustrates these packages and their hierarchy.

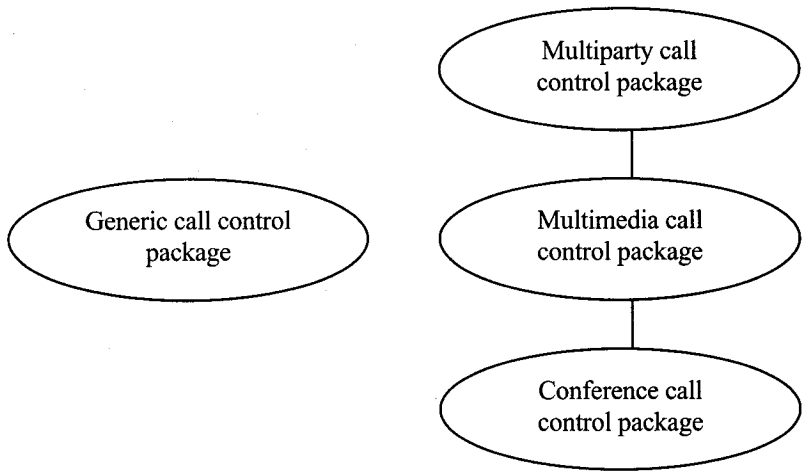


Figure 3.3: Parlay call control packages.

B. Conference Initiation Example

In order to give an example of the methods provided by the call control API, we'll examine the method required to initiate a dial-out conference. The following methods are needed: createConference (); getSubConference (); createCallLeg (); route (); and attachMedia (). createConference () is used to create the conference object. getSubConference () is used to get the list of all the sub-conferences in the conference. createCallLeg (), route (), and attachMedia () are used to create the leg object, route it, and attach it to the conference call. Note that these three methods are called for each participant in the conference. These methods are replaced by the createandRouteCallLeg () method in the latest version of Parlay. This method creates the leg object, routes it, and attaches it, all in one shot. Reference [11] provides more details about the mapping of the Parlay call control APIs onto SIP.

3.1.1.3.3 Pros and Cons

Like JCC/JCAT, Parlay allows the creation of services that are portable across network technologies. In addition, it is programming language neutral and allows the creation of a wide range of services that combine different types of capabilities. Another advantage is that it offers security and resilience features. The main drawback is the low level of abstraction provided. In addition, the use of the Parlay APIs requires a background in circuit switched telephony.

In the context of conferencing applications, the following drawbacks are observed: Due to the low level of abstraction, the number of function calls multiplies rapidly with the increase of the number of participants. This may affect the performance, for a conference with a large number of participants. In the example given in the previous section, to initiate a conference with N participants, 2 + 3N calls are needed. This number is lowered to 2+N calls in more recent versions of Parlay, which is still high. The number of function calls is not an issue in simpler applications, involving a smaller number of users. Another requirement imposed by the application nature is the manipulation of call legs, conference and sub conference objects. These operations require a good knowledge of circuit switched telephony and are not necessarily needed in simpler applications. In addition, the manipulation of multiple call legs and conference objects necessitates keeping tracks of this information, in order to manage the conference.

3.1.2 Signaling Protocol Specific Frameworks

These frameworks are used with specific signaling protocols. H.323 comes with a set of standardized supplementary services and includes no service creation framework. In the

case of SIP, two service creation frameworks exist: SIP Common gateway Interface (CGI) and the SIP servlet API. We now give more details about these two frameworks with a focus on the SIP servlet API. The latter is the most deployed framework in this category.

3.1.2.1 SIP CGI

SIP CGI [21] has been proposed by IETF, targeting experienced and trusted developers. Due to the similarities between HTTP and SIP, the SIP CGI was based on the HTTP CGI [3]. SIP CGI inherits from its predecessor the following characteristics: It is programming language independent and can therefore be used with any programming language. It gives full access to all the messages' headers. It allows the creation of responses and gives access to environment variables. Finally, its similarity to HTTP CGI makes it appealing to web programmers, permitting them to create new services. In the next section, we present the basic operation of SIP CGI and discuss the differences it presents from the HTTP CGI.

3.1.2.1.1 Basic Operation of SIP CGI

A SIP CGI script is first invoked when a SIP request arrives at a server. The server then passes the message body to the script, through its standard input. It also sets some environment variables with information from the headers. The script processes the message body and the environment variables to generate new data. This data is written to the script's standard output then read by the server. The script then terminates.

Several differences exist between SIP CGI and HTTP CGI. First, unlike HTTP CGI, the script output is not necessarily a response message. It can be a proxied request, an entirely new request or even a proxied response. Another important difference is that SIP CGI scripts are persistent while HTTP CGI scripts are not. This property is needed to enable the

script to process subsequent responses to a request that has been proxied or a new request that has been sent. Not all services require the execution of the script for each message received. Therefore, triggering rules have been introduced in SIP CGI to determine the conditions under which a script should execute. Finally, SIP CGI scripts reside on proxy servers to control request routing, while HTTP CGI scripts reside on origin servers.

3.1.2.1.2 Pros and Cons

The main advantage of SIP CGI is that it allows the development of a wide range of services. This is due to the full access to all the headers in addition to the possibility of proxying requests and creating new requests. Another advantage is the programming language independence. Finally, SIP CGI opens service creation to the wide community of web programmers, due to its similarity to HTTP CGI. On the other hand, SIP CGI has several drawbacks. The first drawback is the lack of scalability. SIP CGI scripts reside and run in proxy servers and there is no means to distribute the execution when demand is high. Performance issues constitute another drawback. Despite the introduction of triggering mechanisms to improve the performance, these mechanisms are under specified by IETF. An obvious drawback is the lack of generality, since SIP CGI applies to SIP only.

3.1.2.2 The SIP Servlet API

The SIP servlet API is a standard extension to the Java platform. Its purpose is to standardize the platform for development of SIP based applications, such as conferencing applications in SIP environment. Like SIP CGI, the SIP servlet API targets experienced and trusted developers.

Like the HTTP servlet API [18] from which it is derived, the SIP servlet API builds on the generic servlet API. Despite the similarities between the HTTP servlet API and the SIP servlet API, there exist important differences between them. For instance, SIP servlets reside on proxies rather than on origin servers, in order to control request routing. Apart from the common capability of generating responses, they can initiate requests, receive responses, and proxy requests. The SIP servlet API specification is available at [19].

3.1.2.2.1 The SIP Servlet API Architecture

The SIP servlet API architecture is based on two components: the SIP servlet and the SIP servlet container (or servlet engine). The generic servlet API defines methods that are used in the interactions between servlets and servlet containers. The SIP servlet API is used for the development of the servlets, which are the applications. These applications are run on the container. They extend the capabilities of the SIP server hosting them.

The container can be built into a host SIP server or can be installed as a co-located component. The server hosts a number of applications and contains internal rules determining which application(s) to invoke, based on the messages it receives. The container on the other hand monitors a port on the server where it runs. When the message is received on the port monitored by the container, this entity checks if any servlet-triggering rule has been registered that matches the characteristics of the arriving message. If so, the container loads the servlet identified and executes the service requested. Figure 3.4 illustrates the basic SIP servlet model. In this figure, the container is co-located from the SIP server.

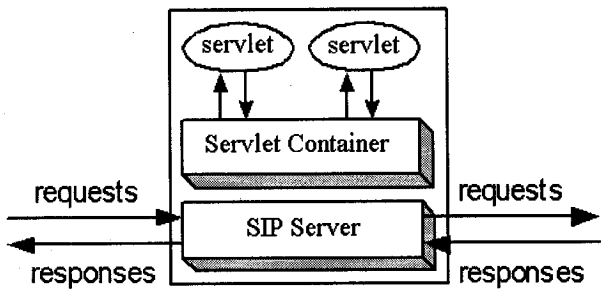


Figure 3.4: The basic SIP servlet model.

The execution of a service is done in three steps: servlet initialization, service execution, and servlet instance destruction. The service execution is controlled using the SIP servlet API. Each request is associated with a method in the API. For instance, if an INVITE request is received, doInvite() method of the servlet is executed. The next section explains how the different requests are handled.

3.1.2.2.2 Processing SIP Messages

When a message is received by the container, this message is forwarded to a servlet instance via the service () method. This method is defined by the generic servlet API. Depending on the nature of the message (request or response), the service () method dispatches it to doRequest () and doResponse () methods. These methods then dispatch further, depending on the type of message. Figure 3.5 illustrates the SIP servlet message handling model.

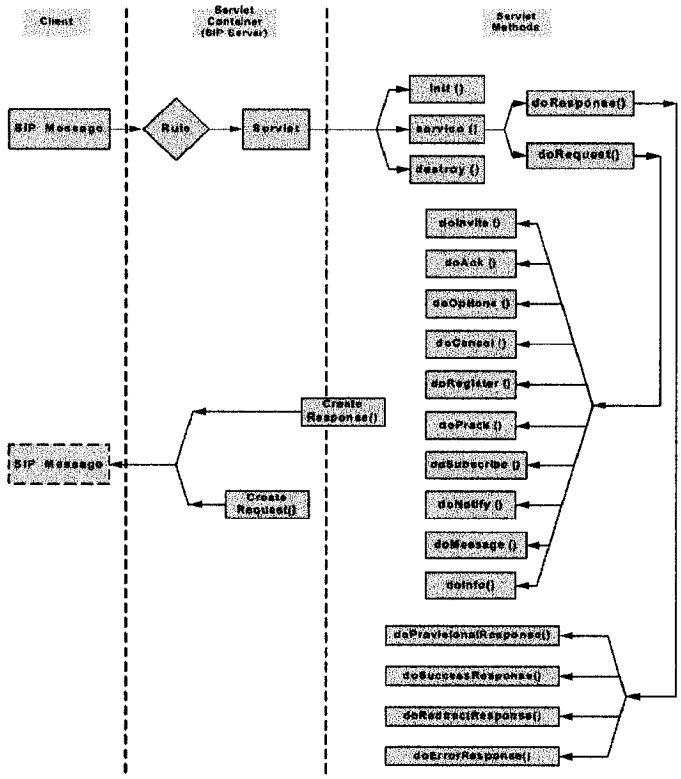


Figure 3.5: SIP servlet message handling model.

A. Request Handling Methods

These methods are part of the SIP servlet API. They are automatically called by the doRequest () method, to help in processing SIP requests. There are ten request handling methods as shown in figure 3.5. The name of each method is related to the request type that it handles. For instance, the doRegister () method is used for handling REGISTER requests, and so on. In the SIP servlet API, the implementation of these methods is empty. This means that they do nothing. Typically, a servlet will override the methods relevant to the service it is providing, and implement them in any way needed. This results in a certain degree of flexibility.

B. Response Handling Methods

These methods are automatically called by the doResponse () method, to help in processing SIP responses. There are four response handling methods: doProvisionalResponse (), doSuccessResponse (), doRedirectResponse (), and doErrorResponse (). The doProvisionalResponse () method is used for handling 1xx responses. 2xx responses are handled by the doSuccessResponse () method, while 3xx responses are handled by the doRedirectResponse () method. Finally, the doErrorResponse () method handles all 4xx, 5xx, and 6xx responses.

C. Access to headers and message body

The SIP servlet API provides some methods allowing the access to the message headers and body. Methods such as getHeader () and setHeader () can be used to access and modify the headers' content. In order to obtain a parsed version of the message body, the getContent () method is provided. For information about all the available methods, the reader can consult reference [19].

3.1.2.2.3 SIP Servlet Operation Example

A simple example to illustrate the model's operation is the call forward on busy service. User A sends an INVITE to user B via the SIP server. The container receives this request, parses it, and identifies the servlet to invoke. It loads the appropriate servlet class and initializes it by calling its init () method. Then, it calls the doInvite () method, passing to it the initial request information as a Servlet Request object. The servlet proxies the request to B, via the container. B sends a busy response, including an alternative address where he can be reached, in the contact header. The container parses the response message and passes its

information to the servlet by calling its doErrorResponse () method. The servlet extracts the new address information from the contact header. It then creates a new INVITE and sends it to the new destination C, via the container. Once the INVITE has been accepted by C, the servlet sends an ACK to it via the container. This way, a connection is established between A and C. When the servlet instance is no more needed, the destroy () method is called by the container. Figure 3.6 illustrates this example.

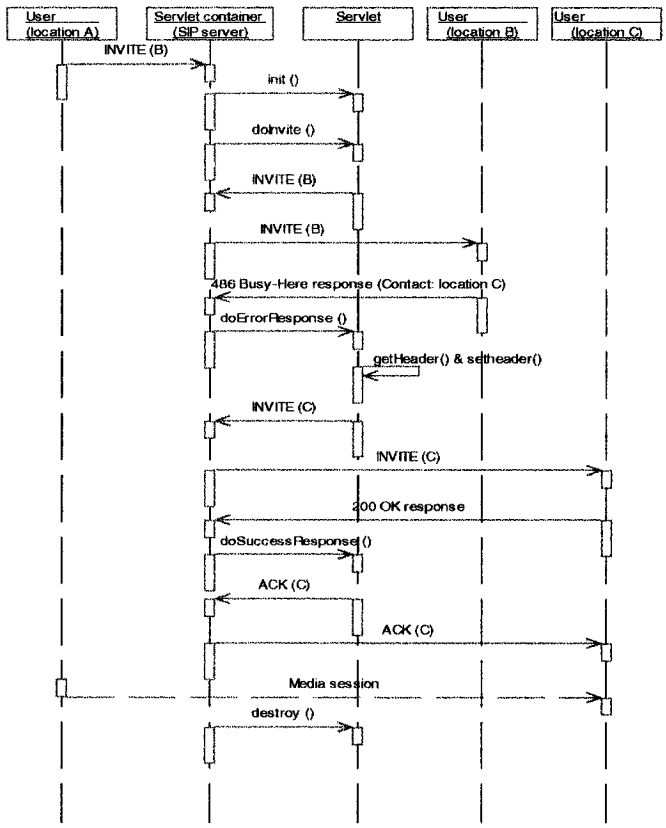


Figure 3.6: An example of a SIP servlet based service.

3.1.2.2.4 Pros and Cons

One of the important advantages of the SIP servlet API is that it allows the creation of a wide range of services. This is due to the full access to all the fields of SIP messages, in addition to the capability of initiating requests, receiving responses, and proxying requests. Two main advantages it has over SIP CGI are scalability and performance. Since the

container and the servlets can be co-located, distributing the execution is possible. Also, due to its similarity to HTTP servlet API, it opens service development to web programmers. Finally, the servlet model offers a certain degree of flexibility. The two drawbacks of the SIP servlet API are its programming language dependence (based on Java) and its lack of generality (specific to SIP).

The use of the SIP servlet APIs for the development of conferencing applications presents the following drawbacks: In order to implement the logic for complex signaling operations such as creating and managing conferences and sub conferences, a deep knowledge of SIP is required. This is due to the extensive manipulation of the SIP messages involved. This level of knowledge is not necessarily required for simpler applications. In addition, a mechanism for keeping track of the participants' addresses, their assigned media ports, the conferences/sub-conferences, and the call-ID assigned to each session is needed. Also, a good knowledge of the servlet model operation is required in order to optimize the design of the servlets used. An optimum design is crucial to reduce the interactions between the servlet and the container, and thus improve the performance. These issues do not arise when dealing with simpler applications.

On the other hand, we chose the SIP servlet API for building our web services gateway for the following reasons: the flexibility offered by the servlet model allows the choice of the appropriate level of abstraction for the gateway; this framework allows the implementation of advanced signaling capabilities such as conferencing and sub-conferencing; this framework is commonly used for the development of signaling capabilities in a SIP environment.

3.2 The Emerging Service Creation Frameworks

In the telecom industry, three main standards are being developed for the adoption of web services: Parlay X, ECMA-348, and the OMA Web Services Enabler (OWSER). The Parlay X web services aim at facilitating applications development by people who are not necessarily experts in telephony or telecommunications. The Parlay X APIs offer a higher level of abstraction than the standard Parlay APIs. Services offered include third-party call, SMS, MMS, payment, account management, user status and terminal location. These APIs focus on two-party calls and do not cover conferencing.

ECMA-348 on the other hand constitutes a complete call control web service specification. Examples of the services defined include capability exchange, routing, and media attachment services. This framework offers an extensive range of capabilities. However, these capabilities are highly specialized for Computer Telephony Integration (CTI) and are not suitable for call control applications in general. In addition, the level of abstraction they offer is still low (WSDL version of existing specifications). Reference [50] can be consulted for the complete listing of ECMA's interfaces.

The OWSER aims at defining common functions in the form of web services. Examples of these functions are: security, network identity, discovery, and service level agreement. Although some of these functions (e.g., authentication) are useful for conferencing, no specific interfaces are defined for call control.

3.3 Conclusions

Several standard service creation frameworks exist for the development of value-added services. These frameworks share a common drawback: they require knowledge which

non-experts may not have. CPL and SIP CGI require the knowledge of scripts, while JCC/JCAT, Parlay, and SIP servlets require the knowledge of their related APIs. In addition, JCC/JCAT and Parlay offer a low level of abstraction and require a background in circuit switched telephony. SIP CGI and the SIP servlet API require knowledge of SIP. These requirements are accentuated for more complex applications such as conferencing applications. In fact, the advanced signaling capabilities used in these applications result in an extensive manipulation of signaling related objects (call legs in the case of JCC and Parlay, message fields in the case of SIP CGI and the SIP servlet API). These capabilities are not possible to implement in a language such as CPL due to the limited behavior of the language primitives.

Emerging service creation frameworks try to remedy to this problem by relying on the web service technology. For instance, Parlay X offers a level of abstraction that is higher than the one offered by the standard Parlay APIs and does not require a background in circuit switched telephony. For the moment, Parlay X web services focus on two party calls and do not address conferencing. ECMA-348 constitutes a complete call control specification including conferencing capabilities. However, the interfaces proposed are still at a low level of abstraction and are rooted in circuit switched telephony. They are therefore not easy to grasp. Finally, OWSER is mainly concerned with common functions such as security and network identify but do not offer any call control interfaces.

The framework we propose uses the same philosophy as these emerging frameworks. It defines comprehensive interfaces exposing conferencing capabilities. These interfaces offer a level of abstraction that is higher than what is offered by the standard service creation

frameworks. In addition, their use does not require any knowledge related to circuit switched telephony or telecom. They can therefore be used by application developers to integrate conferencing capabilities to their applications, easily and rapidly.

Chapter 4

The Case Study Components

In this chapter, we present the different components used in the case study. We then give a thorough description of the function of each component. Among the components presented, we focus on the description of the new interfaces proposed and present the mapping of a sub-set onto SIP. The aim of this chapter is to describe the general operation of the model used in the case study.

4.1 Introduction

The case study was based on centralized conferencing since it is the most appropriate model for Internet telephony applications. Both the dial-in and the dial-out models were considered. The different components involved in the case study are: A conference server which manages the conference from a centralized point and provides web service interfaces exposing its conferencing capabilities. The gateway, along with the servlets and the mixer play the role of the conference server in our model. Another component is the application which uses the capabilities exposed by the interfaces. Finally, the SIP clients are SIP user agents representing the application users. Note that each client has a point-to-point signalling and media relation with the conference server. This chapter provides more details about these components and figure 4.1 illustrates them.

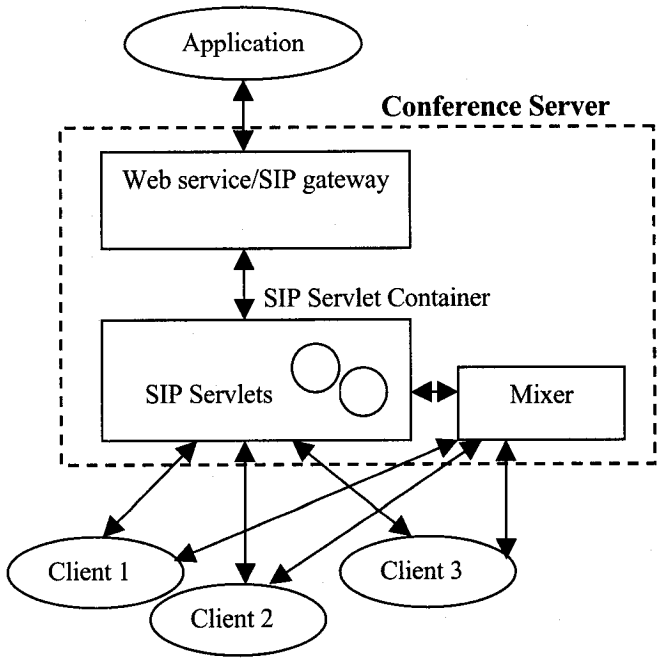


Figure 4.1: The case study components.

We note that the dial-in and the dial-out models use two different modes of communication. The dial-out model uses synchronous communication while the dial-in model uses asynchronous communication. In synchronous communication, the client application sends a request to the conference server and remains blocked until it receives the response. This type of communication is suitable for dial-out conferences since all actions are initiated by the application and sent to the server, which directly gets the result and sends it back to the application. However, this mode is not suitable for dial-in conferences since the application will remain blocked for a long time before a user joins or an update occurs, to receive a response. Asynchronous communication solves this problem by allowing the application to subscribe to a certain event and get notified when the event occurs. Two notification mechanisms can be used: the callback mechanism and the polling mechanism. We now give a brief overview of both mechanisms. Figure 4.2 illustrates the synchronous mode and the asynchronous mode (using the two notification mechanisms). More information about the two notification mechanisms can be found in [2].

In asynchronous communication using callbacks, the application first sends its request to the conference server and directly gets an acknowledgement for it. This terminates the first request-response interaction. When the final result of the operation is ready, the server sends it to the application via a callback. The callback is a request (not a response) sent to the application which replies with an acknowledgement. This constitutes the second request-response interaction.

When using the polling mechanism, the application once again sends its request to the server and directly gets an acknowledgement. Afterwards, it starts polling the server regularly to check if its request has been completed. When the request is completed, the application calls the server to get the result.

The callback mechanism seems to be an efficient way to implement asynchrony. In fact, only one additional response-request interaction is needed to send the notification. For that reason, we chose to implement the asynchronous mode of communication used by the dial-in model, based on the callback mechanism.

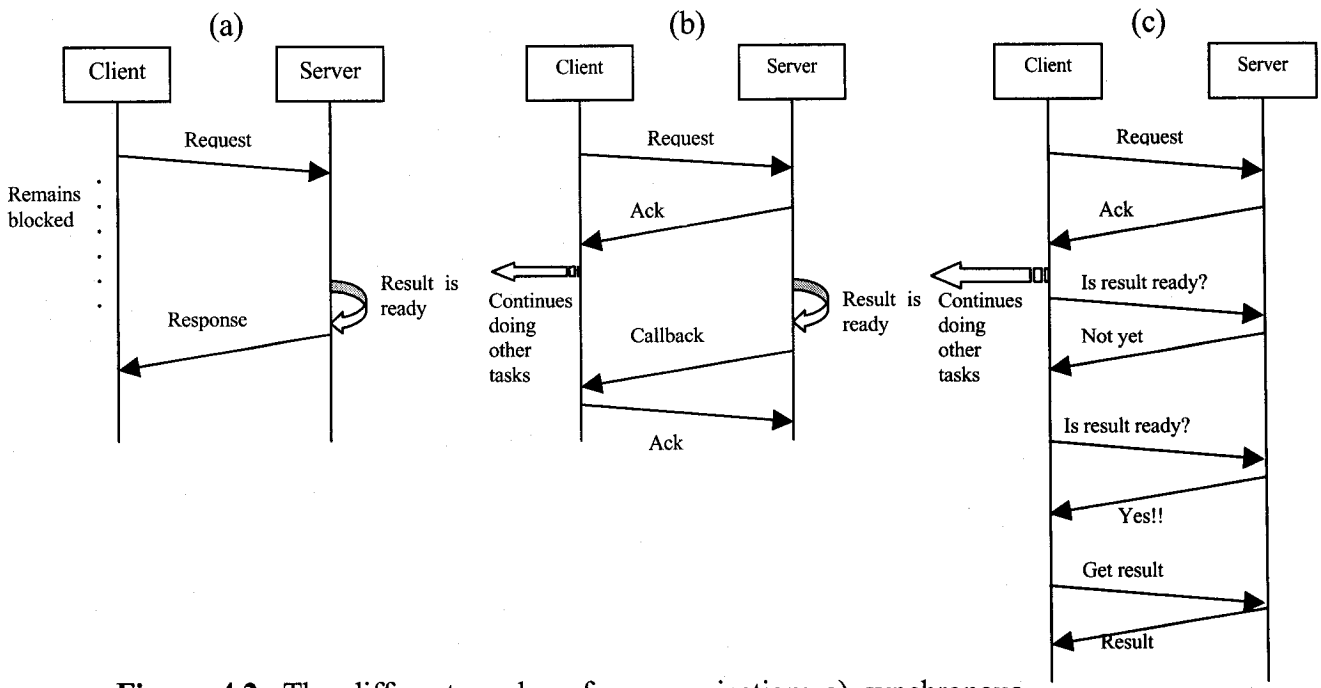


Figure 4.2: The different modes of communication: a) synchronous communication; b) asynchronous communication using callbacks; and c) asynchronous communication using the polling mechanism.

4.2 The Gateway and the Servlets

The gateway and the servlets are key elements in the model studied. They are responsible of implementing the interfaces the applications expect to find, and mapping the applications' calls onto SIP messages going to the clients. In this section, we first present the interfaces defined. We then discuss the mapping of a sub-set of the interfaces onto SIP.

4.2.1 The New Web Service Interfaces

The new interfaces proposed define conferencing related functions which suit the needs of both dial-in and dial-out conferencing models. In this section, we describe the functions specific to each model. We then present the functions common to both models. A summary of the defined functions is presented in table 4.1.

4.2.1.1 Methods Specific to the Dial-out Conferencing Model

Eight methods were defined to provide conferencing and sub-conferencing capabilities to the dial-out model. These methods are: initiateConf (); addUser (), removeUser (); initiateSubConf (); endSubConf (); moveUser (); splitSubConf (); and mergeSubConf (). All these methods are implemented by the gateway, except the splitSubConf () and the mergeSubConf () methods. These two methods were not implemented since the applications developed did not involve splitting and merging of sub-conferences. The dial-out audio conference uses the full range of the implemented functionalities.

initiateConf () is used to initiate a dial-out conference using the parameters provided. It takes as input the following parameters: the addresses of the participants to invite to the conference; the media type supported by the conference (presently only voice is supported); the conference duration (in seconds); the expected number of participants; and the conference ID (input-output parameter updated with the conference ID generated). It gives as output the result of the conference initiation (successful or not).

addUser () and removeUser () are used to add/remove a participant to/from the conference after it has started. Both methods take as input the address of the participant to add/remove and the ID of the conference concerned. Both return the result of the operation.

initiateSubConf () is used to create a sub conference using the parameters provided. The sub conference will be related to the main conference already started. initiateSubConf () takes the following input parameters: The addresses of the participants to move into the sub conference after its creation; the ID of the main conference to which the sub conference will be related; and the sub conference ID (input-output parameter to be updated with the

sub conference ID generated). This method doesn't trigger any signaling action. However, media manipulation is performed. We explain the media action involved in the next section. The result of the sub conference initiation is returned as output.

endSubConf () is used to terminate the specified sub conference. The sub conference ID and the main conference ID are input parameters. The result of the sub conference termination is returned as output.

moveUser () is used to move a participant between two sub conferences or between a sub-conference and the main conference. The input parameters are: the address of the participant to move; the main conference ID; the ID of the sub conference of origin; and the ID of the sub conference of destination. To move a participant from the main conference to a sub conference, the origin sub conference ID should equal zero. In the opposite case (moving from a sub conference to the main conference), the destination sub conference ID should equal zero. Like initiateSubConf (), moveUser () doesn't trigger any signaling action. It only involves media manipulation. This method returns the result of the operation as output.

splitSubConf () is used to create a new sub conference and move some of the participants to it. The input parameters are: the main conference ID, the ID of the sub conference of origin; the addresses of the participants to move; and the ID of the sub conference of destination (input-output parameter to be updated with the destination sub conference ID generated). The result of the sub conference splitting is returned as output. This function does not trigger any signaling action. However, it involves media manipulation.

mergeSubConf () is used to merge two sub conference together, moving all the participants from the sub conference of origin to the sub conference of destination, then destroying the sub conference of origin. The input parameters are: the main conference ID; the ID of the conference of origin; and the ID of the conference of destination. Once again, this method requires media manipulation only. It returns the result of the sub conferences merging as output.

4.2.1.2 Methods Specific to the Dial-in Conferencing Model

In this case, two methods were defined specifically for the dial-in model. These methods are: initiateAsyncConf () and getConfUpdate (). Both methods are implemented by the gateway. They are used by the dial-in audio conference.

Contrarily to its dial-out counterpart, initiateAsyncConf () does not trigger any signaling action. It is used to set a dial-in conference room waiting for the participants to join in. It takes the same input parameters as initiateConf (), except for the participants' addresses since they are not known yet. This method directly returns an acknowledgement to the application. When the first user joins in, the updated conference information is returned to the application in a callback. This mechanism describes the asynchronous mode of communication used for dial-in conference initiation.

getConfUpdate () is used to investigate about any updates occurring during the conference. It takes the conference ID as input parameter. Like initiateAsyncConf (), getConfUpdate directly returns an acknowledgement to the application. When an update occurs (a user joins or leaves the conference), the updated conference information is returned to the application in a callback. Thus, this method uses asynchronous communication.

4.2.1.3 Methods Common to Both Models

Four methods were defined to be used by both models. These methods concern conference termination (both dial-in and dial-out) and floor control capabilities. Only the endConf () method was implemented by the gateway. The remaining methods were not implemented since they were not needed for the applications developed. Note that resource reservation methods were not defined since they would represent a low level of abstraction and would require circuit switched telephony background to understand their purpose. Therefore, we decided to omit them from the interfaces defined. We now present a description of the common methods defined.

endConf () is used to terminate the specified conference. It takes as input the ID of the conference to terminate and returns the result of the conference termination. In the dial-out case, it is used to terminate a dial-out conference when the chair decides to end the conference. In the dial-in case, it is used to terminate a dial-in conference when the time allowed expires.

requestFloor () is used to inform the application that a participant requests the right to use the microphone. The input parameters are: the main conference ID; the ID of the sub conference to which the participant belongs; and the participant address. If the participant does not belong to any sub conference (only to the main conference), the sub conference ID should be set to zero. The application can then grant the requested floor by calling the appointSpeaker () method. No output is returned by the requestFloor () method.

appointSpeaker () is used to appoint a floor (e.g., the right to use the microphone) to the indicated participant. The input parameters are: the main conference ID; the ID of the sub

conference to which the participant belongs (zero if does not belong to any sub conference); and the participant address. This method returns the result of the operation as output.

appointChair () is used by the application to select a participant as the conference chair. The input parameters are: the main conference ID; the ID of the sub conference to which the participant belongs; and the participant address. This method returns the result of the operation as output.

Method signature	**Related Model**	**Description**
Boolean **initiateConf**(String[] addresses, String mediaType, int duration, int expectedUsers, String confID);	Dial-out model	Used to initiate a conference with the given users' addresses. The conference ID is an output parameter returned.
boolean **addUser**(String userAddress, String confID);	Dial-out model	Used to add/remove a user to/from a conference. The user's address and the conference ID must be specified.
Boolean **removeUser**(String userAddress, String confID);		
boolean **initiateSubConf** (String[] users, String ConfID, String subConfID);	Dial-out model	Used to initiate a sub-conference between the specified users. The main conference ID to which they belong must be specified.
Boolean **moveUser** (String user, String ConfID, String OriginSubConfID, String DestinationSubConfID);	Dial-out model	Used to move a user between sub-conferences or main conference and sub-conferences and vice-versa.
boolean **splitSubConf** (String[] users, String ConfID, String OriginSubConfID, String DestinationSubConfID);	Dial-out model	Used to split a sub-conference into two, by moving some of the participants of the first sub-conference to a second new sub conference.
Boolean **mergeSubConf** (String ConfID, String OriginSubConfID, String DestinationSubConfID);	Dial-out model	Used to merge two sub-conferences into one, by moving the participants of the first sub-conference to the second then destroying the first sub-conference.
boolean **endSubConf**(String ConfID, String subConfID);	Dial-out model	Used to terminate a certain sub-conference. Both main and sub-conference IDs are used.

void **initiateAsyncConf** (String mediaType, int duration, int expectedUsers, String confID);	Dial-in model	Used to set a dial-in conference room, waiting for the users to join in. (Asynchronous mode)
Void **getConfUpdate** (String confID);	Dial-in model	Used to investigate about any updates occurring during the conference.(Async. mode)
boolean **endConf**(String confID);	Dial-out and Dial-in models	Used by application to terminate a conference of the given ID.
void **requestFloor** (String userAddress, String ConfID, String subConfID);	Dial-out and Dial-in models	Used to inform the application that a participant is requesting a floor (e.g., the right to use the mic.)
boolean **appointSpeaker** (String userAddress, String ConfID, String subConfID);	Dial-out and Dial-in models	Used by application to grant a floor request to a participant.
boolean **selectChair** (String userAddress, String ConfID, String subConfID);	Dial-out and Dial-in models	Used by application to select a chair for the conference.

Table 4.1: The new web service interfaces.

4.2.2 Mapping the Application Function Calls onto SIP

As mentioned previously, only a sub-set of the functions presented were implemented. These functions were mapped onto SIP. Two levels of mapping are performed. First, each of the functions implemented is mapped onto a SIP message by the gateway. This message is sent to the SIP container which interacts with the appropriate servlet to perform the needed signaling to the conference clients. Therefore, the servlet performs a second level of mapping, transforming the message received by the container into SIP messages sent to the clients. The two levels of mapping were performed using core SIP messages, in addition to one extension method (the INFO method). In this section, we first present the mapping strategy, and then detail the mapping of the implemented functions.

4.2.2.1 The general Mapping Strategy

SIP does not offer conference control services. Instead of introducing new SIP extensions to support conferencing needs, we tried to accommodate those needs using basic SIP

messages. Four possible solutions were examined to accomplish this task:

1. Use the trivial solution of sending a SIP message to each of the users, via the SIP servlet (hosted by the container), which serves as a proxy in this case.
2. Incorporate the users' addresses as SIP URLs in several contact headers and the remaining information (ID, action to take) in the subject header. These headers are then accessed by the servlet, which takes in charge the signaling of the users.
3. Insert the conference parameters in the message body. This body is then parsed by the servlet to access the needed information and perform the signaling actions.
4. Use messages pointing to a multicast address (representing an administrative group to which the users should belong).

Despite its simplicity, the first solution is very inefficient and was therefore discarded. The second solution has the advantage of accommodating any number of users (number of contact headers used is unlimited). However, only REGISTER messages can accommodate multiple contact headers. Since the REGISTER message does not reflect the nature of all the functions to map, this solution was discarded. The last solution is more related to the multicast architecture not the centralized architecture used. Finally, the third solution is the one we chose since it is intuitive and accommodates a fairly large number of users. In addition, it could be applied with several SIP messages, reflecting the nature of the functions to map.

Therefore, the mapping strategy consists of the following: At a first stage, the gateway maps each function to a specific SIP message which reflects its nature. Each SIP message contains the needed conference parameters in its message body. The SIP message is then passed to the servlet which performs a second mapping, transforming it into other SIP messages needed for the clients signaling. Figure 4.3 illustrates the mapping strategy. This strategy is detailed below:

initiateConf () and addUser () were mapped onto INVITE messages. The INVITE message naturally reflects the action of inviting a user (or users) to a session (in this case, a conference session). The servlet then maps this INVITE message onto other INVITE message(s) sent to the client(s).

endConf (), removeUser (), and endSubConf () were mapped onto BYE messages. The BYE message reflects the action of terminating a session with a user (or users). Again, the servlet maps this BYE message onto other BYE message(s) sent to the client(s).

initiateSubConf (), moveUser (), initiateAsyncConf () and getConfUpdate () were mapped onto INFO messages. The INFO message was chosen in those cases, to reflect actions that do not require any signaling. In fact, those four functions involve media manipulation or resource initialization. Therefore, the INFO message is used to inform the servlet of a media action request or resource initialization request. No further mapping is required for the INFO messages received by the servlet.

getConfUpdate () does not require any action from the servlet. It consists of a database query request to the gateway itself. Since no SIP message need to be sent to the container, no mapping is required for this function.

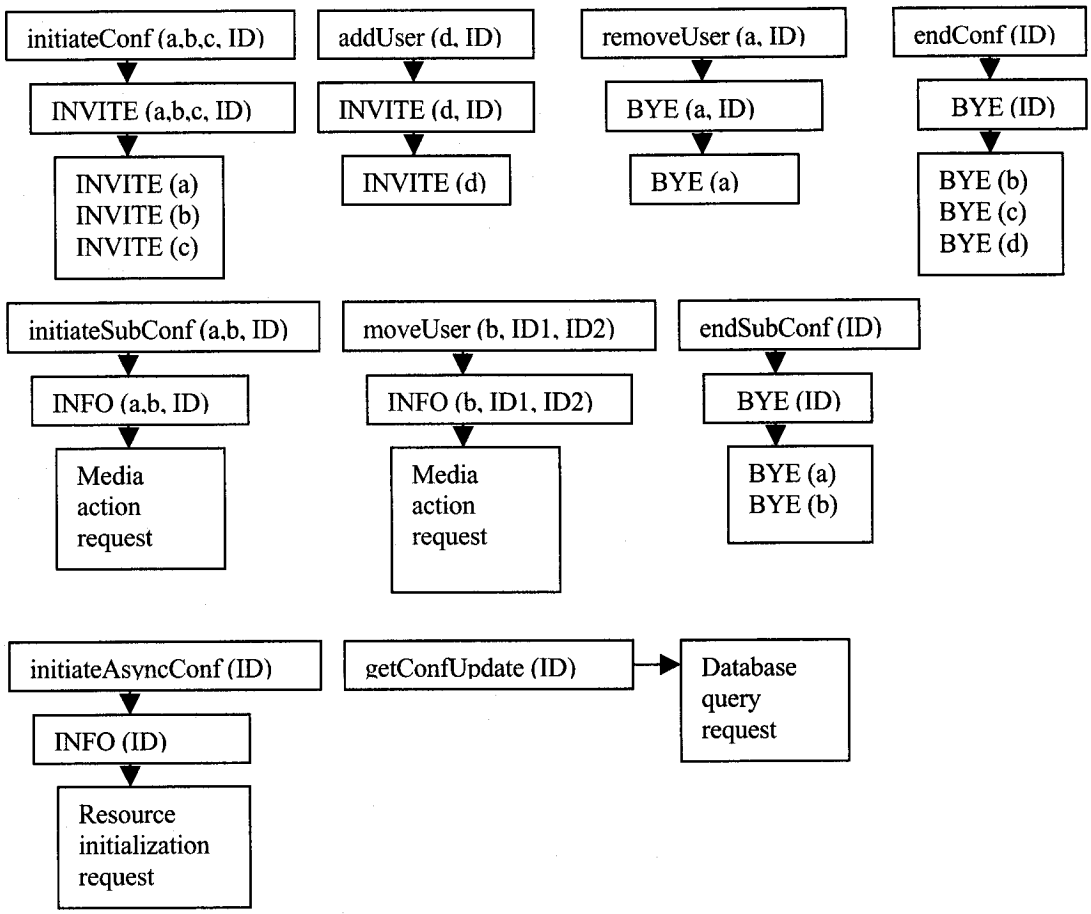


Figure 4.3: The Mapping Strategy.

4.2.2.2 Detailed Mapping of the Dial-out Functions

initiateConf () is first mapped onto an INVITE request by the gateway. This request contains the needed information (users' addresses and conference ID) in its message body. It is then sent to the SIP container which parses it and identifies the servlet to invoke. The container loads the appropriate servlet class and initializes it by calling its init () method. Then, it calls the doInvite () method, passing to it the initial request information as a Servlet Request object. The servlet extracts the conference information using the getContent () method. It then creates new INVITE messages (one for each participant), and sends them to the SIP clients via the container. Once the invitations have been accepted by the clients, ACK messages are sent to them by the servlet, via the container. The servlet then calls one

of the mixer's functions (addStream ()) to add the users' media streams. Finally, the servlet updates its users' database and sends an INFO message to the gateway via the container. This message provides information about the conference initiation. This message is mapped by the gateway to a return parameter, which is returned to the application.

addUser () is first mapped onto an INVITE request by the gateway. This request contains the needed information (action requested, user address, and conference ID) in its message body. As described previously, the SIP request is sent to the SIP container, which forwards it to the servlet by calling its doInvite () method. Once again, the servlet parses the message body to extract the needed information. It then creates a new INVITE message and sends it to the user to be added, via the container. Once the user accepts the invitation, an ACK message is sent to him by the servlet (via the container). The servlet calls the mixer's addStream() method to add the user stream. It then updates its users' database and sends another INFO message to the gateway to inform it of the result of the function requested. This message is then mapped by the gateway to a return parameter, sent to the application.

removeUser () is first mapped onto an BYE request by the gateway. This request contains the needed information in its message body. It is forwarded by the container to the appropriate servlet (via the doBye () function), which extracts the needed information. It then creates a new BYE message and sends it to the user to be removed from the conference. Once it is accepted by the user, the servlet calls the mixer's subtractStream() function to subtract the user stream. The rest of the scenario is the same.

The endConf () is first mapped onto a BYE request sent by the gateway. This request contains the conference ID in its message body. This message is sent to the container which

forwards it to the servlet by calling its doBye () method. The servlet extracts the needed information and creates new BYE messages. These messages are sent to the remaining conference participants, via the container. Note that the servlet keeps track of the participants in a database. Once those participants accept the BYE requests, the servlet calls the mixer's subtractStream() function to subtract their streams. It then clears its users' database and sends an OK response to the gateway, via the container. This response is mapped by the gateway to a return parameter, sent to the application. This method is shared by both models.

Figure 4.4 illustrates the mapping of the functions presented. Note that the interactions between the container and the servlet are not shown.

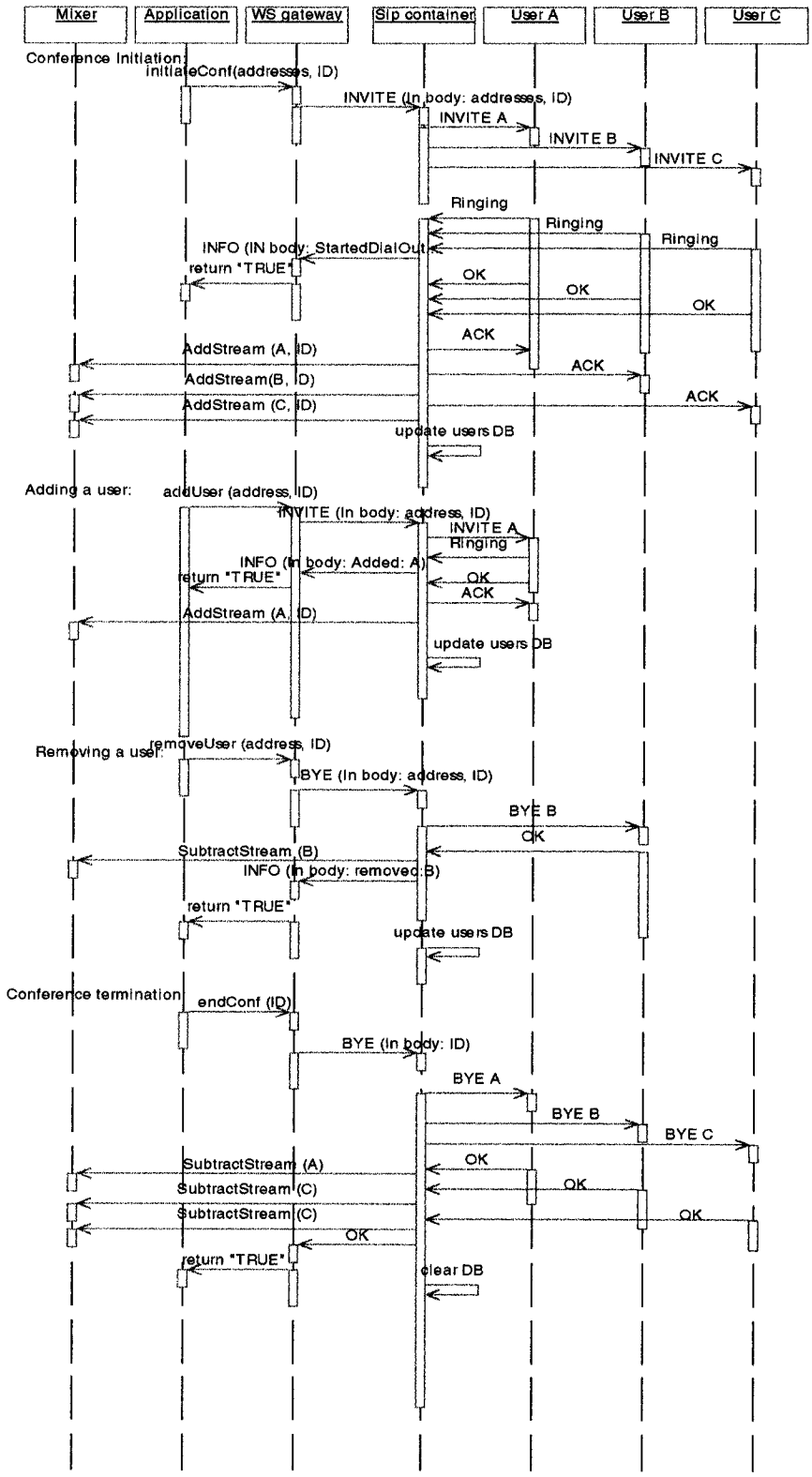


Figure 4.4: Dial-out mapping: conferencing functionalities.

initiateSubConf () is first mapped onto an INFO request sent by the gateway. This request contains the needed information (action, users' addresses, and sub conference ID) in its message body. As described previously, the SIP request is sent to the SIP container, which forwards it to the servlet by calling its doInfo () method. The servlet extracts the needed information. It then calls the mixer's moveStream () function to move the users' streams from the main conference stream pool to the sub conference stream pool. Note that no signaling action is needed here, since signaling relations already exists with all the users (due to the main conference initiation). The servlet then updates its users' database and sends another INFO message to the gateway.

endSubConf () is also mapped onto a BYE request sent by the gateway. This request contains the action and sub conference ID in its message body. This message reaches the servlet via the doBye () method. The servlet extracts the needed information and creates new BYE messages. These messages are sent to the sub conference participants, via the container. When the participants accept the BYE requests, the servlet calls the mixer's subtractStream() function to subtract their streams. It then updates its users' database and sends an INFO message to the gateway.

moveUser () is mapped onto an INFO request sent by the gateway. This request contains the needed information (action, user address, main conference ID, origin and destination sub conferences IDs) in its message body. Once the servlet extracts the needed information, it then calls the mixer's moveStream () function to move the user stream. Once again, no signaling action is required. The servlet then updates its users' database and sends an INFO message to the gateway.

Figure 4.5 illustrates the mapping of the sub conferencing related functions.

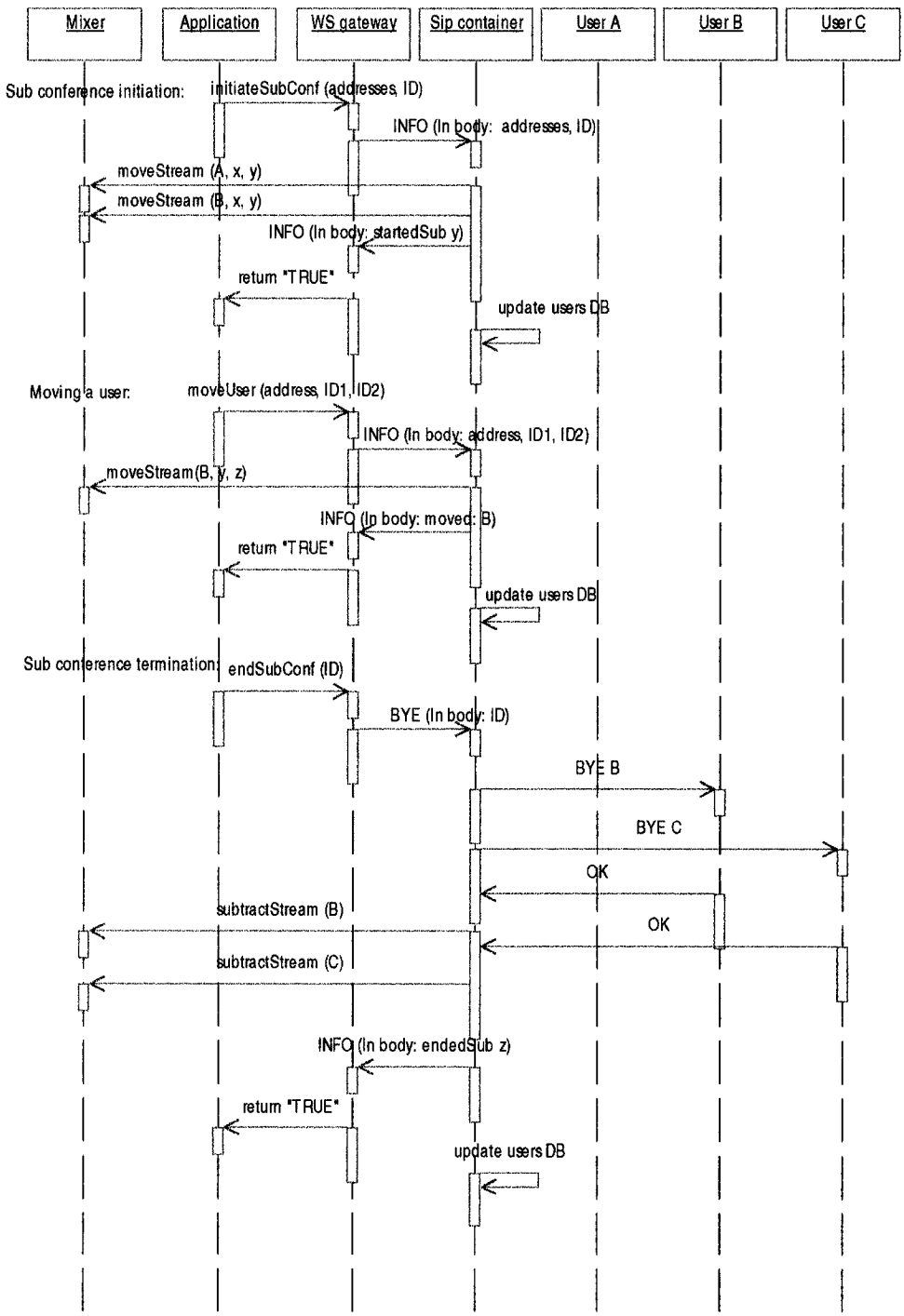


Figure 4.5: Dial-out mapping: sub conferencing functionalities.

4.2.2.3 Detailed Mapping of the Dial-in Functions

When the initiateAsyncConf () function is called, the gateway directly sends an acknowledgement to the calling application. This terminates this first request-response interaction. The initiateAsyncConf () function is then mapped onto an INFO request by the gateway. This request contains the conference ID in its message body. It is then sent to the SIP container which forwards it to the dial-in servlet by calling its doInfo () method. The servlet extracts the conference ID and sets a conference room, by initializing the needed resources. Later on, when the users start sending their invitations to join the conference room, the servlet finalizes the signaling and contacts the mixer to handle the media mixing. After the first user joins in, an INFO message is sent back to the gateway to inform it of this event. This information is used to update the conference record, within the gateway, which sends it via a callback to the application. The callback in this case is a request, not response. The application finally responds with an acknowledgement.

When the getConfUpdate () function is called, the gateway directly sends an acknowledgement to the calling application. No SIP messages are sent to the container as a result of this function. In fact, each time an update occurs, the servlet automatically sends an INFO message to the gateway to inform it of the update. The information in these INFO messages is used to update the conference record maintained by the gateway. When the getConfUpdate () function is called, the gateway examines the conference record and check if it has been changed from the last getConfUpdate () function call. If it has changed, the new conference information is sent to the application via a callback. The application then responds with an acknowledgement.

Figure 4.6 illustrates the dial-in functions' mapping.

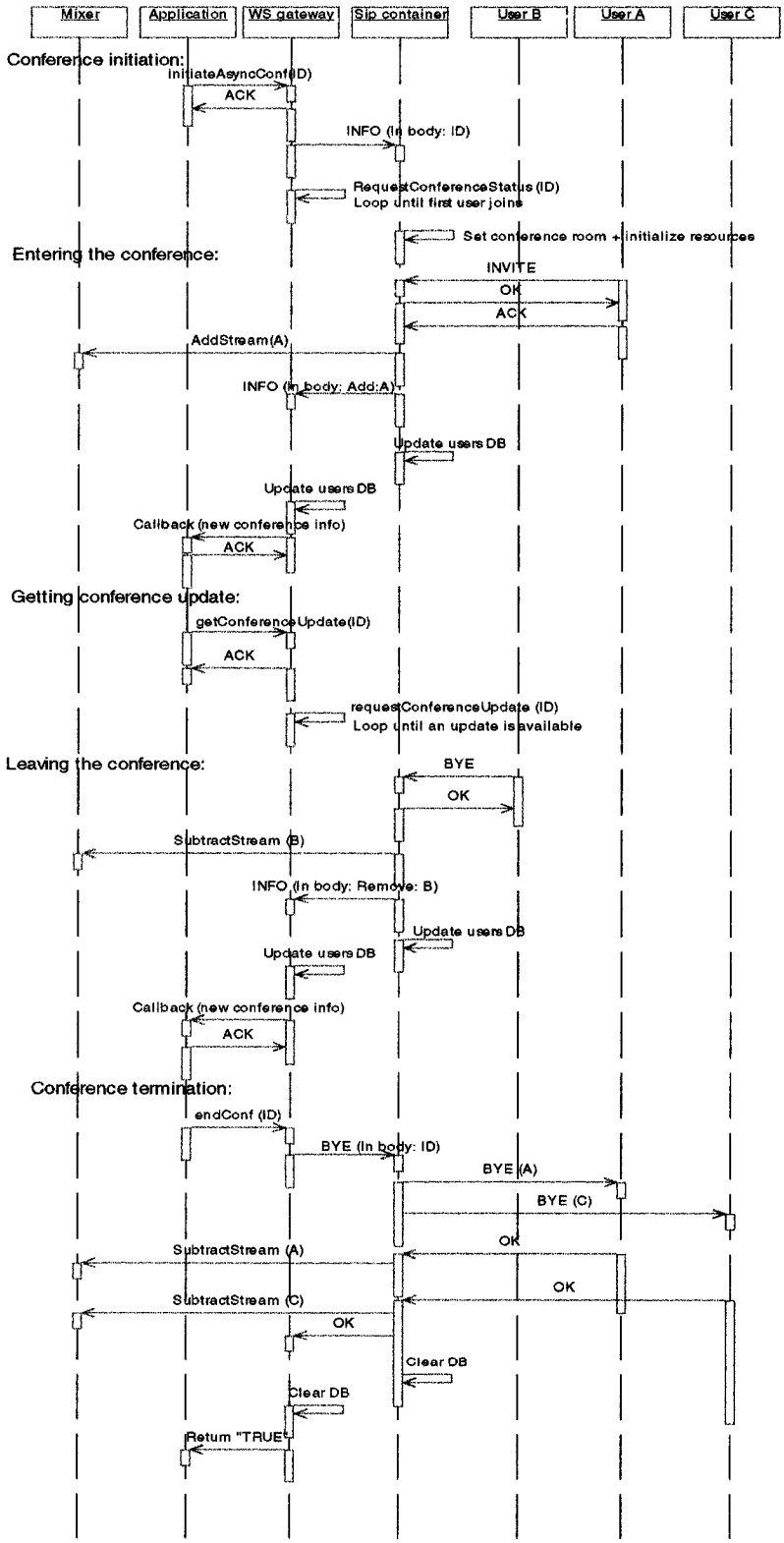


Figure 4.6: Dial-in mapping.

4.3 The Mixer and the SIP Clients

The Mixer is a support unit required for conferencing. It handles media mixing by enabling actions such as: the addition of a user stream to a pool of streams; the subtraction of a stream from a pool of streams; and the moving of a stream between two pools of streams.

The SIP clients are the conference participants. They are normal SIP user agents, able to send and receive SIP messages. They also receive mixed streams from the mixer. In the dial-out case, they can only accept invitations. In the dial-in mode, they can initiate invitations and terminate sessions as they wish.

4.4 The Client Applications

Several applications could be built using the capabilities of the conferencing model presented. Two audio conferencing applications were implemented as examples: a dial-in and a dial-out audio conferencing application. Some features of these applications are described as follows: Both conferencing applications are pre-arranged. We chose to study floor control by including it in one of the applications - the dial out conference. In this case, a chair conducts the conference, orchestrating the access to the shared resources. Furthermore, the dial-out conference includes sub-conferencing capabilities, such as creating sub-groups and moving users between sub-groups. The dial-in conference does not include these capabilities, as they are already included in the dial out model. The dial-out conference is closed, so users cannot join the conference as they please. The dial-in conference on the other hand is open for anyone to join. Finally, the dial-out conference is started and ended by the chair. The dial-in conference is started when the first user joins in, and it ends when the time allowed expires. In the dial-in case, other conditions for the

starting and ending of the conference could have been used. These features are summarized in table 4.2.

Dial-out Conference Features	Dial-in Conference Features
▪ Pre-arranged ▪ Conducted by chair (with floor control) ▪ Dial-out ▪ Sub-conferencing capabilities ▪ Closed ▪ Join allowed ▪ Media negotiation ▪ Only chair can invite user. Users can't join. ▪ Chair starts conference and ends it as he wishes.	▪ Pre-arranged ▪ Non-conducted (without floor control) ▪ Dial-in ▪ No sub-conferencing capabilities ▪ Open ▪ Join allowed ▪ No media negotiation ▪ Users can join and invite other users ▪ Conference is launched automatically with first comer. System ends conference when time allowed expires.

Table 4.2: Audio conferences features.

Chapter 5

Implementation and Performance Measurements

This chapter first presents the components software architecture and the prototypes developed. Afterwards, performance evaluation is discussed.

5.1 Implementation

Among the components presented, the two applications, the gateway, and the servlets were built during the implementation part of this research work. The SIP container, the mixer and the SIP clients were built previously by other groups and were reused in this project. In this section, we focus on the conference server (gateway, servlets, and mixer) since it is the key component. The applications were built for demonstration purposes and their architectures are out of the scope of this thesis. For more details about the reused components, references [15, 9] can be consulted.

5.1.1 The Software Architecture of the Conference Server

As shown in figure 5.1, the conference server is composed of two units: a signaling control unit and a media control unit. The gateway and the servlets (hosted by the container) act as the signaling control unit. This unit performs the actual mapping between the calls made to the web service interfaces and SIP. The mixer on the other hand represents the media control unit. The latter is a support unit required for conferencing. It is responsible of media mixing. The communication between the signaling unit and the media unit is done using Java calls.

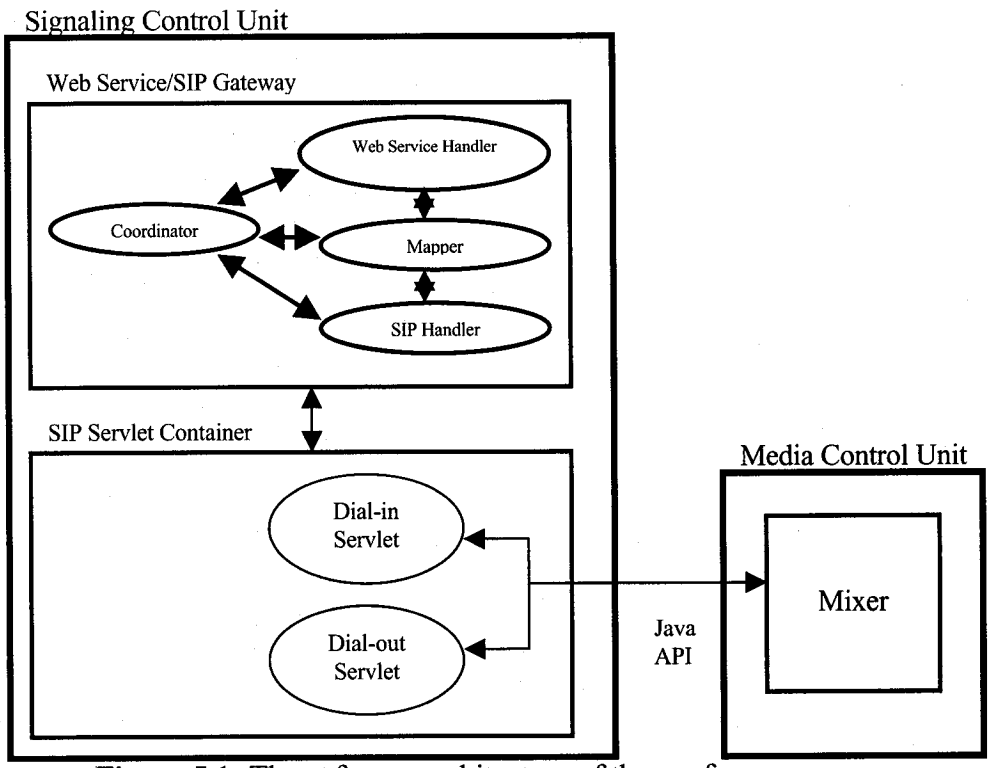


Figure 5.1: The software architecture of the conference server.

The signaling control unit has three interfaces: HTTP, SIP, and Java API (for communication with the media control unit). It is composed of two elements: the gateway and the servlets hosted by the SIP container. The gateway implements the interfaces the applications expect to find and maps the applications calls onto SIP messages sent to the container. The container forwards each SIP message it receives to the appropriate servlet. Two servlets were designed to suit the needs of both dial in and dial-out models. The container uses servlet-triggering rules to direct the messages to the appropriate servlet. For instance, all messages with the word "dial-in" in their subject headers are directed to the dial-in servlet. The servlet then maps the message it receives onto other SIP messages sent to the clients.

5.1.1.1 The Modules and their Roles

The gateway is composed of four modules: the web service handler; the mapper; the SIP handler; and the coordinator. The web service handler validates the SOAP messages received from the applications and builds the SOAP messages to be sent to the applications. SOAP messages are carried over HTTP. The SIP handler builds and sends SIP messages to the container which forwards them to the appropriate servlet to handle the clients' signaling and interactions with the mixer. The SIP handler also validates the SIP messages received from the container. The mapper handles the mapping between the web service handler and the SIP handler. The coordinator keeps track of all actions and handles any exceptions or errors. It also manages the interaction with the conferences database. This database contains conferences' records.

When the web service handler receives a valid SOAP request, it extracts from it the name of the function required and the parameters to use. It then determines if mapping is required. If it is, it asks the mapper to perform the mapping. The mapper then invokes the SIP handler, which builds and sends the appropriate SIP message to the container. The container forwards this message to the appropriate servlet. The servlet extracts the name of the function required and the parameters to use from the message received. It then determines if a second mapping is required. If it is, it builds and sends the new SIP messages to the SIP clients, via the container. The servlet also manages the interaction with the mixer. In addition, it processes the responses it receives back and sends feedback messages to the gateway, via the container.

Upon receipt of a SIP message from the container, the SIP handler validates it and sends it to the mapper. The mapper determines if mapping is required. If so, it does the mapping and invokes the web service handler. The web service handler builds the corresponding SOAP message. In the case of asynchronous communication, the web service handler uses the appropriate callback function to notify the application. In the case of synchronous communication, the handler sends the resulting message as a normal SOAP response.

We note that the way the subscription-notification cycle is accomplished in asynchronous mode is not optimized. Normally, an application could subscribe for a number of events by calling a certain function (ex: getConfUpdate ()→ notify me each time a user joins or leaves). The application would then receive a notification for each of these events. However, due to a limitation imposed by the application server that we used, only one notification could be sent per subscription. Therefore, when the application calls the getConfUpdate () function, it will get notified through a callback as soon as the first update occurs. To get the other updates, other calls to the getConfUpdate () function must be made by the application.

The media control unit has two interfaces: Java API (for communication with the signalling control unit) and RTP (Real Time Protocol). RTP is the protocol used in Internet telephony for media handling. This unit is composed of a simple media mixer. A more complicated design with a media manager and media handlers could have been used for the support of several media types. The mixer was designed by another group; therefore its architecture will not be discussed in this thesis. The mixer receives and validates the RTP media streams from the conference participants. It mixes them and redistributes the resulting RTP

streams to the appropriate participants. We chose to connect the mixer with the servlet and not the gateway for synchronization purposes. In fact, when the servlet triggers both clients' signalling and media handling, synchronization between the two operations can be achieved.

5.1.1.2 The Class Diagram

Figure 5.2 shows the class diagram. It illustrates the modules presented and their related methods and attributes.

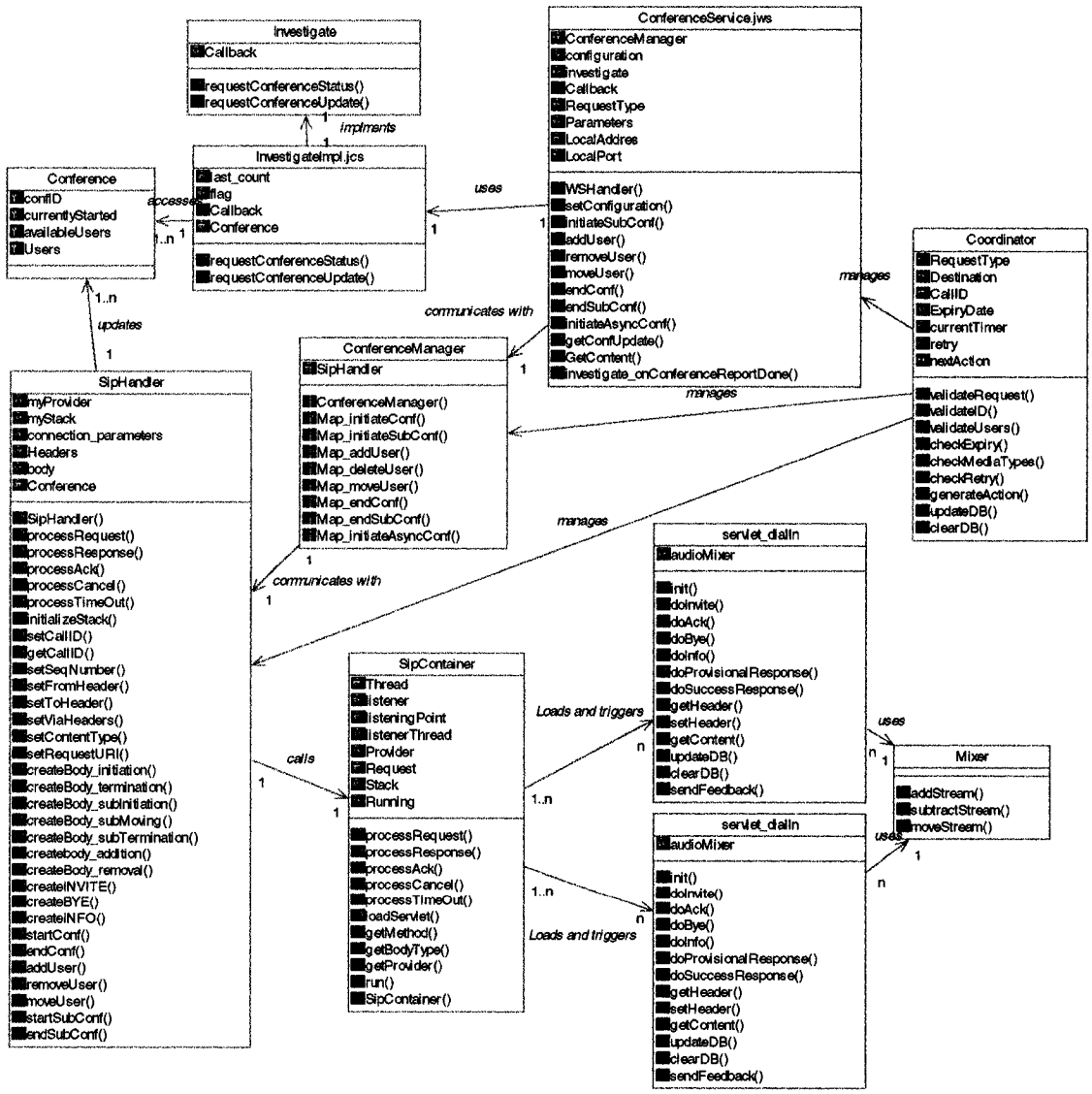


Figure 5.2: The class diagram.

5.1.2 The Prototypes

Figure 5.3 shows the overall architecture of the dial-out conferencing application prototype. The prototype is made of an application server, a conference server, and SIP clients. A similar prototype was built for the dial-in conferencing application.

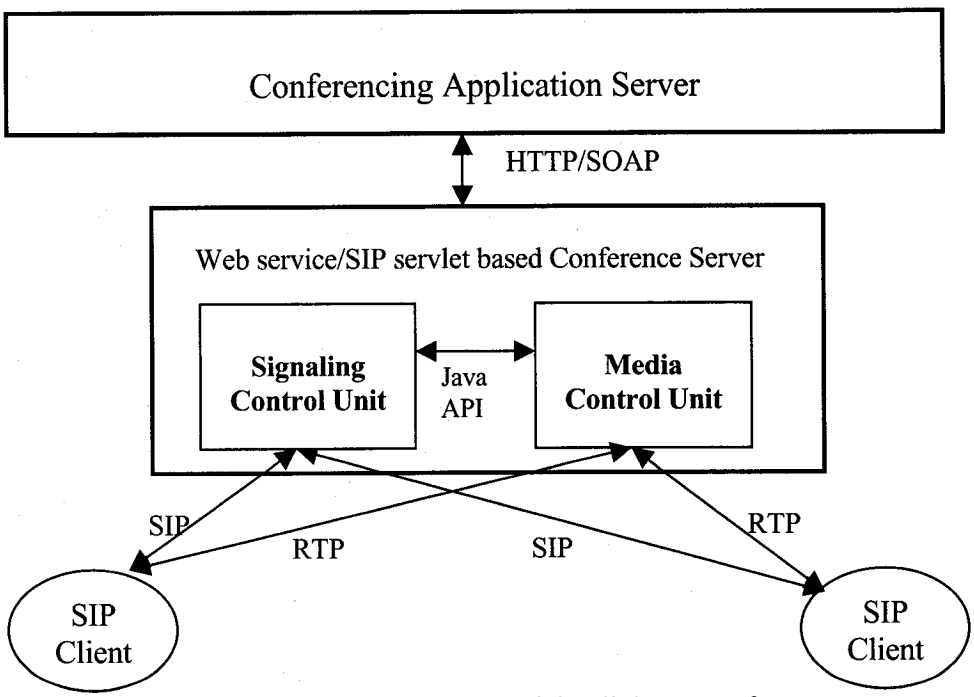


Figure 5.3: Overall architecture of the dial-out conference prototype.

5.1.2.1 The Components Implemented

In the signaling control unit, the gateway and the two servlets have been fully implemented. Instead of a full-blown SIP container, we used a standalone SIP container providing basic functionality. The media control unit was fully implemented in another project. A PointBase database was created to store information about the conferences. The prototypes were implemented in Java. Each of the components presented was implemented as a separate unit for scalability purposes. For instance, another SIP container, or media unit can be used by the gateway implemented, since all the components are decoupled.

We used the Java API provided by the mixer in the communication between the signaling and the media control units. This API provides functions such as: addStream (); subtractStream (); and moveStream (). These functions allow respectively: the addition of a user stream to a conference/sub-conference; the removal of a user stream from a conference/sub-conference; and moving a user stream from a sub-conference to another.

5.1.2.2 The Web Services Deployment

The interfaces provided by the conference server were deployed on BEA Weblogic 8.1 application server. This resulted in two web services: a dial-in conferencing web service and a dial-out conferencing web service. WSDL documents were generated to provide descriptions of the web services. These documents were used for the generation of stub classes necessary for the invocation of the web services. Both applications were fully implemented and bonded to the appropriate web services using the stub classes generated. Figure 5.4 illustrates the different steps for the web services' deployment.

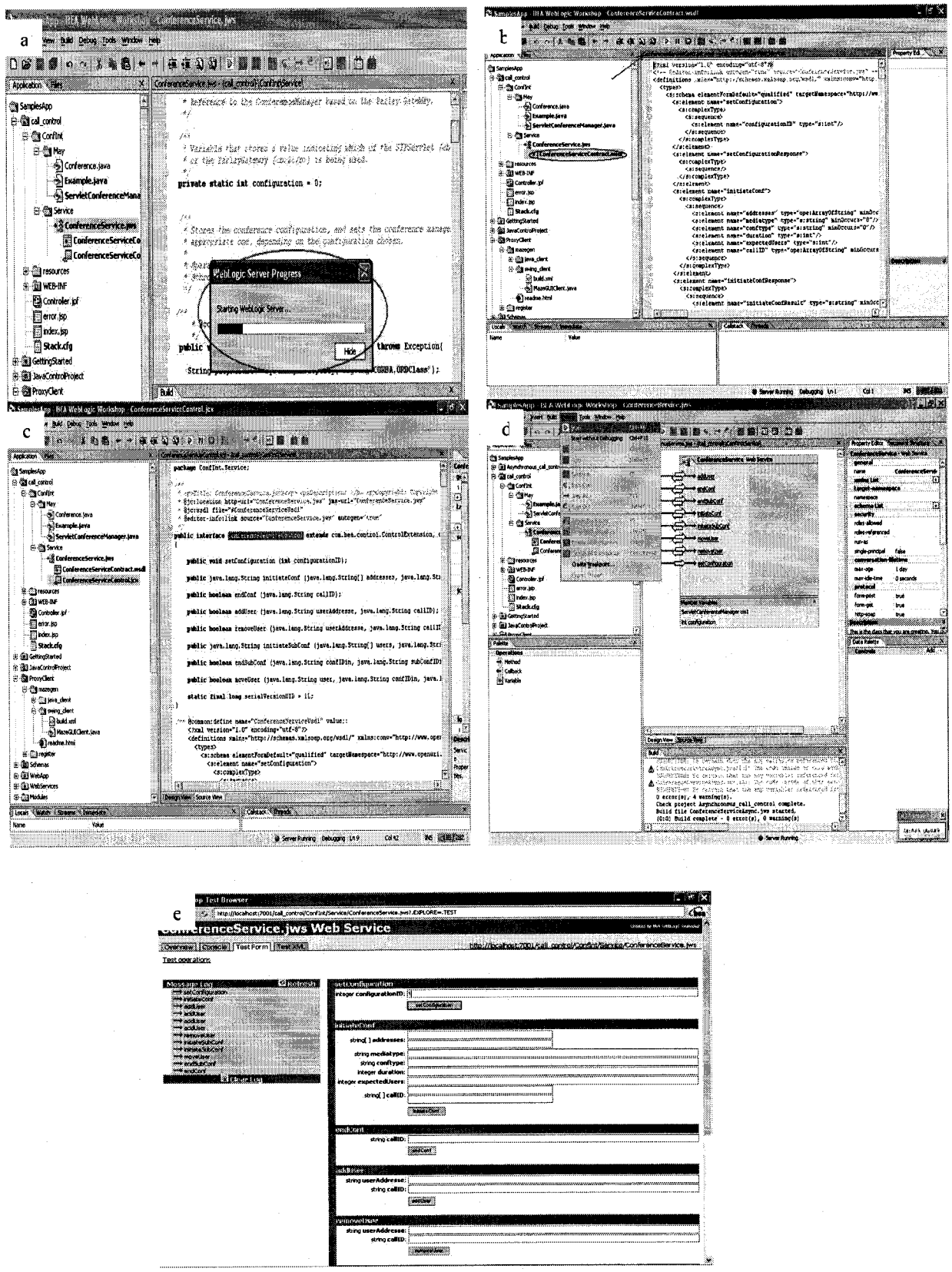


Figure 5.4: Web services deployment steps: a) server starting; b) WSDL document generation; c) stub classes generation; d) web service building; e) web service testing.

5.1.2.3 Asynchronous Mode Requirements

Two requirements were met in order to be able to support callbacks when operating in asynchronous mode. There requirements are:

- The web service defining a callback must be conversational. This means that the web service must be able to keep track of the originator of a request, and can therefore send the callback to the appropriate caller. This is accomplished, by establishing a unique conversation session between the server and each application it communicates with. Each conversation, having a unique identifier, permits the conference server to direct the callbacks to the appropriate application. BEA application server allows the creation of conversational web services. Therefore, this condition was satisfied in the dial-in conferencing web service deployed. Figure 5.5 illustrates a conversation example.
- The application itself must be capable of receiving and interpreting a callback. Since the callback is defined by a web service, then the application must also be a web service, to be able to receive SOAP requests (not only SOAP responses). In addition, the application must also be capable of correlating an incoming message with a previous request that it initiated. Finally, the application should not be protected by a firewall, which would consider the callback as an unsolicited request and would block it. These conditions were satisfied in the dial-in audio conference application. The latter is the only application operating in asynchronous mode.

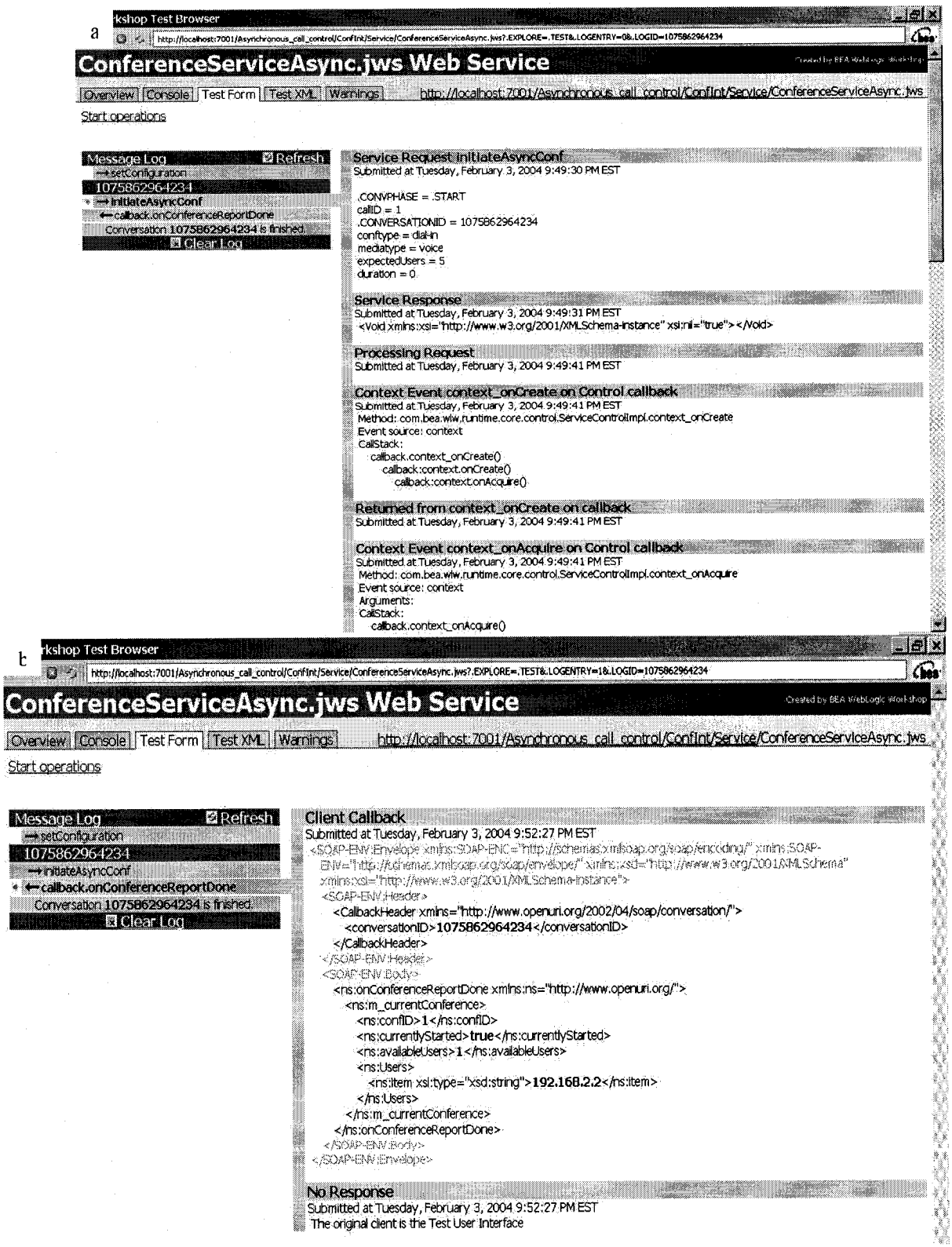


Figure 5.5: Example of conversational interaction with the dial-in web service: a) initial request; b) callback.

5.1.2.4 The Setup

The SIP container, the servlets and the mixer were installed on the same machine due to the frequent communications between them. The interactions are carried locally and do not induce any extra load on the network. The workstation is a 400 MHz, 600 RAM, Intel Pentium II machine running Windows 2K. The gateway is installed on a separate machine running BEA Weblogic 8.1 application server. It is a Pentium IV machine operating at 2.2 GHz with 512 RAM and running Windows XP. The same applies to the server sides of the applications (audio conference servers) and the applications' clients (SIP clients). Five SIP clients were used as participants to the conferences.

5.1.2.5 A Word about the Interfaces

The main advantage of the interfaces implemented is the higher level of abstraction they provide. This level of abstraction could allow application developers who are new to the telecom field, to easily incorporate call control functionality in their applications. Another advantage is the loose coupling between the interfaces and the applications developed. This loose coupling allows changes in the interfaces implementation without necessitating any change in the applications using them. Finally, the use of these interfaces allows a considerable reduction in the application foot print. In fact, using the traditional SIP servlet approach requires 740 lines of codes to incorporate the presented dial-out call control capabilities in an application. Using the web service interfaces requires only 70 lines of code. These figures have been observed in the dial-out voice conferencing application developed. This important reduction of the application foot print saves time and effort for application developers.

5.2 Performance Measurements

The measurements were taken at night due to the absence of a dedicated and isolated network. Taking the measurements at night allows the minimization of obstructions and other hindering factors. The measurements were taken for the dial-out model only since the dial-in model results depend on the participants' reaction time and are therefore not relevant. For instance, the dial-in conference initiation time depends on when all the participants decide to join the conference. The dial-out audio conferencing application was used for testing. All workstations are connected to a 100 Mb/s Ethernet LAN segment. This section first presents the metrics and the data collected. The data is then analyzed.

5.2.1 Metrics and Measurement Data

Two metrics were used: time delay and network load. The time delay was calculated as the time duration between a function call and a feedback message is received from the container concerning the operation result. Note that for conference initiation and user addition, the feedback message is sent when a ringing is received from the participant (or the last participant) to be invited. The network load was calculated as the size of packets exchanged to accomplish a certain function. The packets were captured using Ethereal software [8]. The time of capture of each packet was used in the calculation of time delays. Both metrics are influenced by the number of participants targeted.

In addition to the measurements made for the web service interfaces, similar measurements were taken for the raw SIP servlet model. The comparison of these two batches of measurements was used to calculate the overhead introduced by the web service layer. Figure 5.6 shows the two models compared.

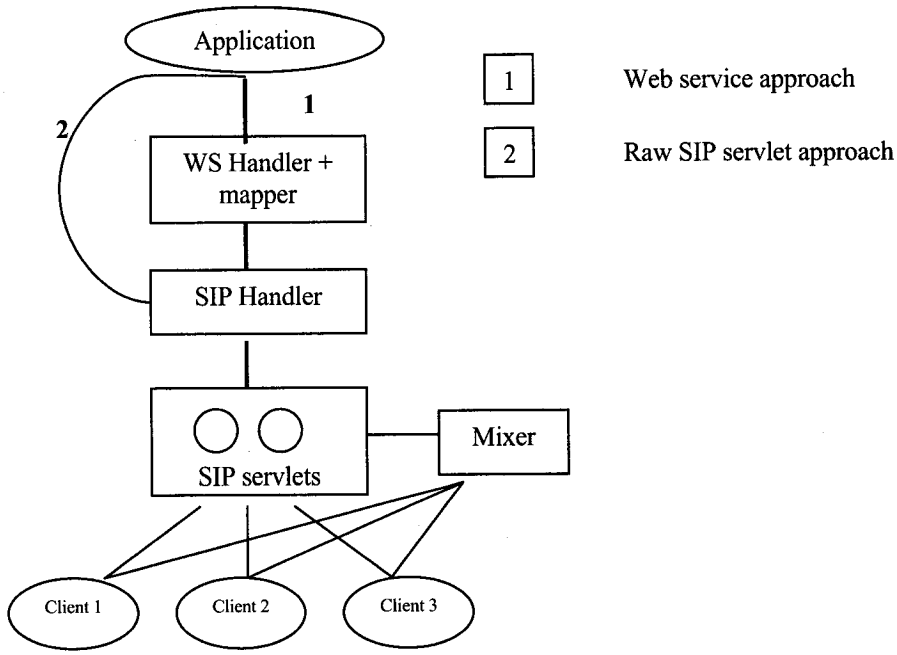


Figure 5.6: Web service approach Vs raw SIP servlet approach.

For each approach, three sets of measurements were taken. These sets involve targeting three, four, and five participants respectively. Each set consists of 15 trials. For each trial, the dial-out conference is initiated and its full functionality tested using the dial-out audio conference application. The following sequence of actions was used: The conference is initiated with all the participants available (3, 4 or 5 participants); a participant is removed then added again; One sub conference is created with two random participants; a participant is moved from the sub conference to the main conference; the sub conference is ended (dismissing only one participant); and the main conference is ended (dismissing the remaining participants). The following conditions were observed: The SIP clients are online at the conference initiation time. They always accept invitations made to them (no busy or No-answer).

For each approach, the time delay for each of the seven functionalities is plotted against the number of targeted participants. The overhead introduced by the web service approach is shown in a separate graph. The same is done for the network load. Two additional graphs are also presented: the time delay measurements and the network load distribution for a four-participants-conference, using the web service approach.

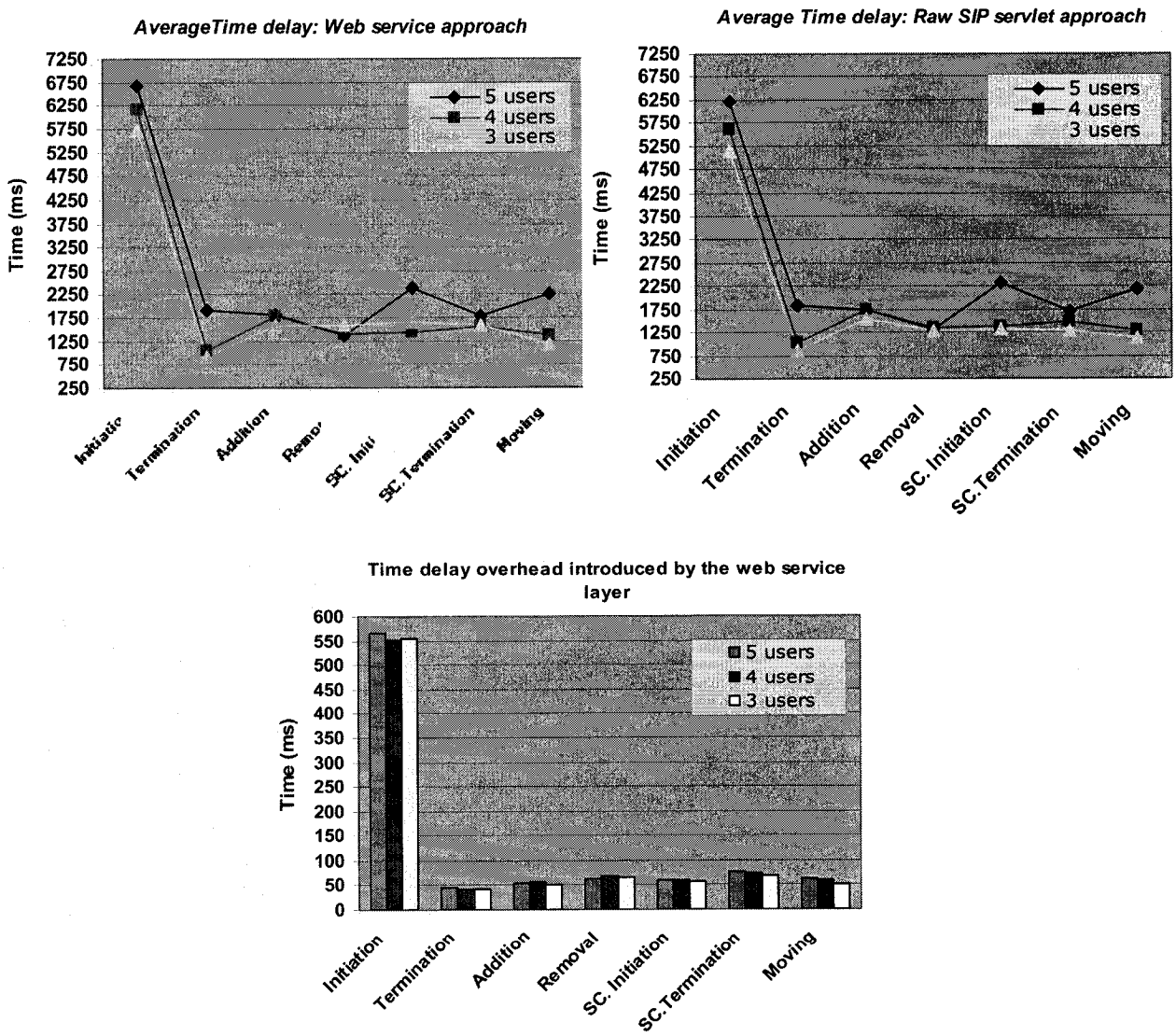


Figure 5.7: Average time delay and time delay overhead: a) average time delay for dial-out functionality using the web service approach; b) average time delay for dial-out functionality using the raw SIP servlet approach; c) time delay overhead generated by the web service layer.

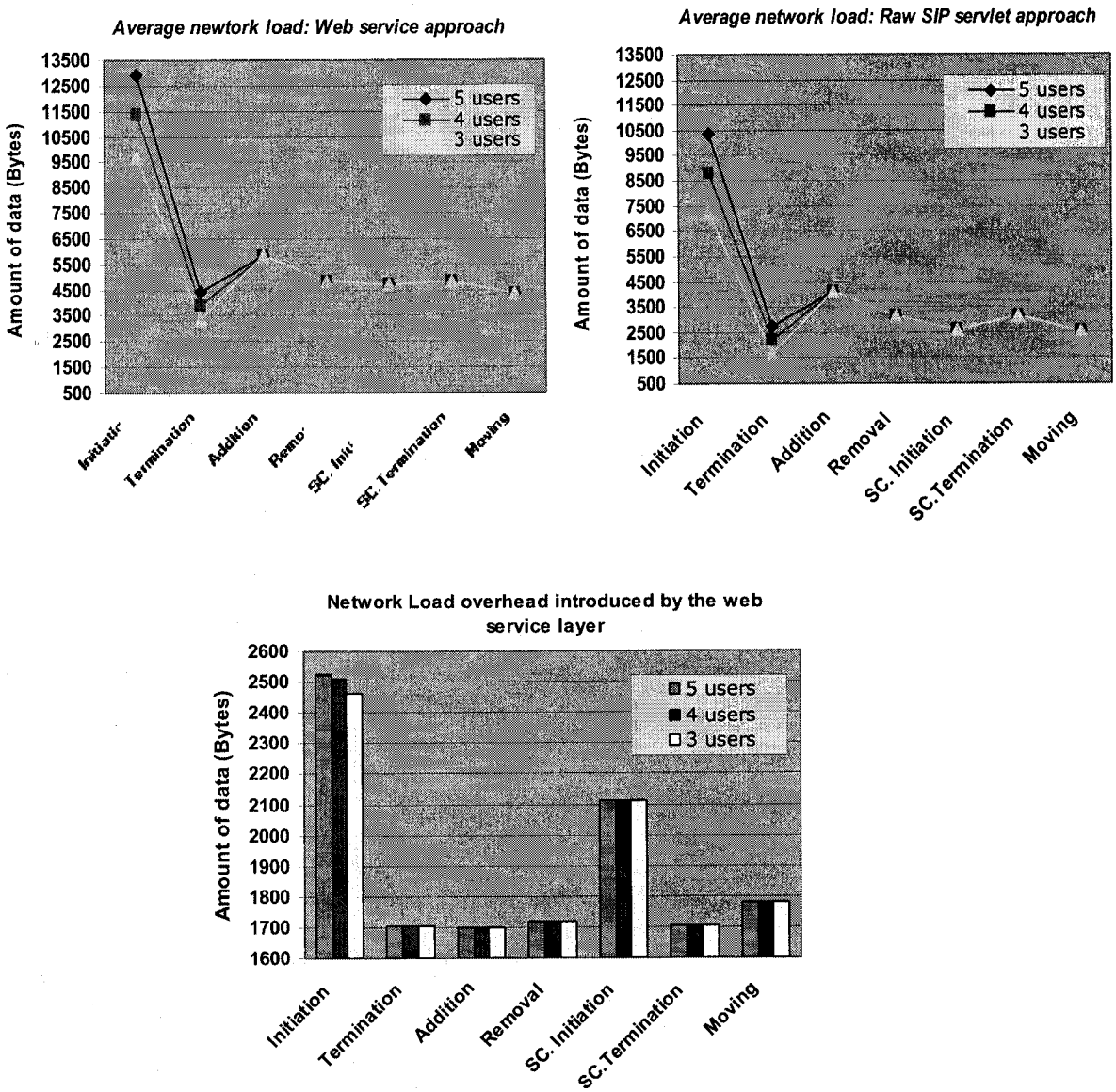
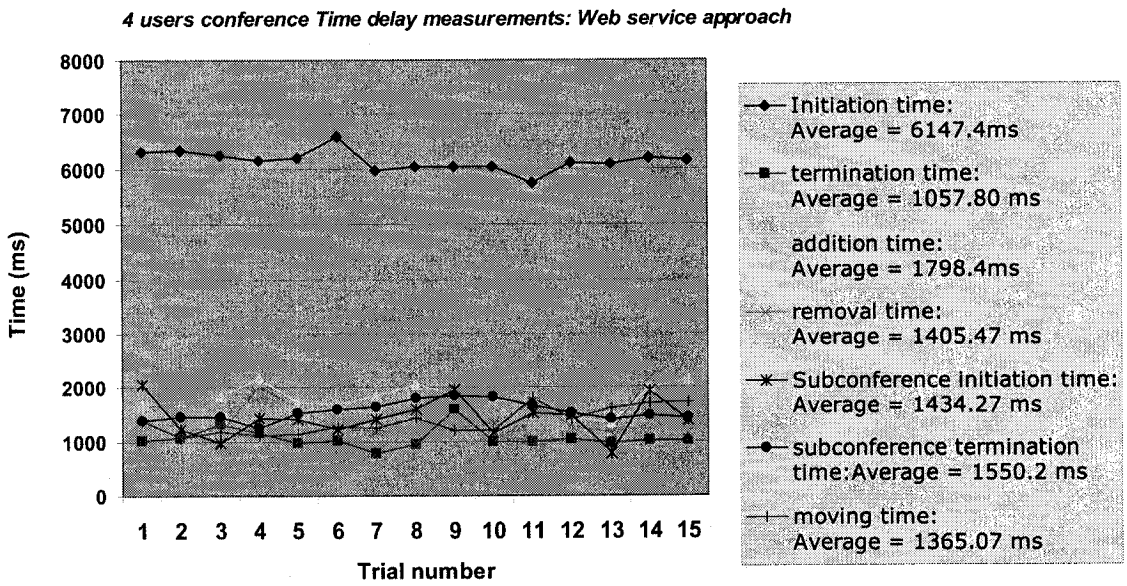


Figure 5.8: Average network load and network load overhead: a) average network load for dial-out functionality using the web service approach; b) average network load for dial-out functionality using the raw SIP servlet approach; c) network load overhead generated by the web service layer.



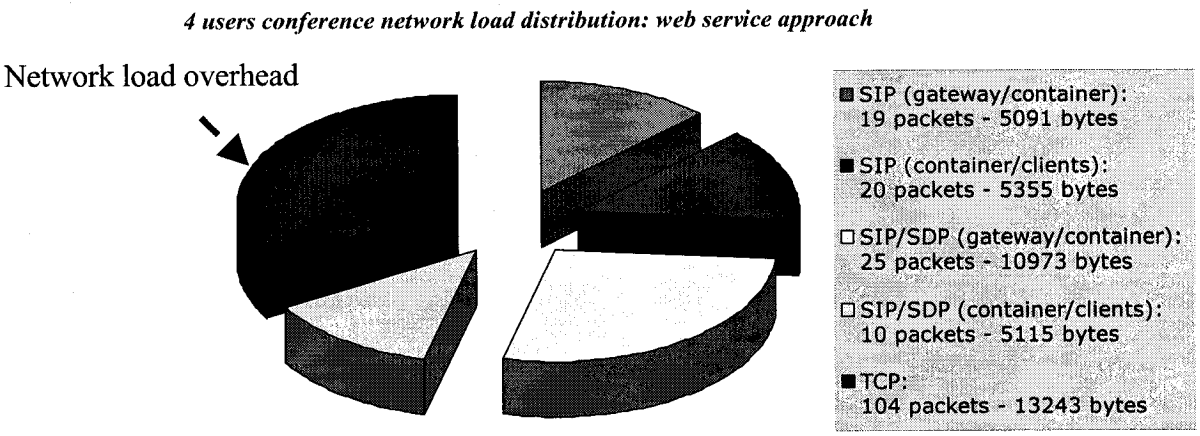


Figure 5.9: Time delay trend and network load distribution: a) time delay measurements for a four-participants-conference (web service approach); b) network load distribution for a four-participants-conference (web service approach).

5.2.2 Time Delay Data Analysis

As shown in figs 5.7a and 5.7b, the biggest delay is generated at conference initiation (5-7 seconds). This delay is caused by several factors: servlet loading time, initialization of needed resources, and users' location in the network. The average dial-out servlet loading time has been calculated to 119 ms. We note that a servlet instance is loaded only once, at the conference initiation. The same instance is then reused for the remaining operations. The same applies for resource initialization and users' location. Therefore, the delays for the other operations are much less (1-2 seconds).

Initiation and termination time delays are proportional to the number of participants. This is due to a linearly proportional relation between the number of participants and the number of SIP messages exchanged. We note that the initiation/termination delays variation is not linear since other operations (coordination and database manipulation) are involved. The remaining operations are slightly affected by the number of participants. These operations involve signaling to one client or no signaling at all. The slight variations observed are related to differences in database manipulations.

As expected, the web service approach introduces an overhead in terms of time delay. This overhead is due to the extra communication between the application and the web service layer. Once again, the biggest overhead is introduced at conference initiation, when the first TCP connection is established to carry the traffic between the application and the web service. We notice that the overhead is almost constant with respect to the number of participants (fig 5.7c). In fact, only one SOAP message is sent per function, irrespective to the number of participants involved. The slight variations result from the different delays

while establishing the different TCP connections. The average overhead ranges between 43 and 556 ms (3% – 10 % respectively). This delay is barely felt by the application's user. Therefore, the added web service layer does not significantly penalize the performance of the system.

Finally, the time delay trend is uniform as shown in figure 5.9a. In fact, no big fluctuations are observed in the performance. We note that the SIP container is kept running without restarting during the total number of trials. Therefore, we can conclude that the SIP container used did not present any garbage collection problems.

5.2.3 Network Data Analysis

As shown in figs 5.8a and 5.8b, initiation and termination network loads are linearly proportional to the number of participants. Other operations loads are constant with respect to the number of participants. These operations generate the same amount of traffic in any case, since they always target one participant or do not generate any signaling packets (relative to participants).

The network load overhead generated by the web service layer consists of the extra SOAP messages exchanged between the application and the web service. The packets size varies in the case of the conference initiation, depending on the parameters passed within the message. Therefore, the increase in the number of participants is reflected in the number of parameters, thus increasing the overhead generated. For the remaining operations, the number of parameters was kept constant for all scenarios. Therefore, the overhead generated by these operations is unaffected by the number of participants. Figure 5.8c illustrates the network load overhead generated.

The average overhead per functionality ranges between 1.7 and 2.4 Kb (41% - 28% respectively). Even though this overhead is considerable, it remains constant with respect to the number of participants, for almost all operations. Other solutions to decrease the SOAP messages weight should be investigated.

In the web service approach, three types of packets are exchanged during the communications between the different components: SIP and SIP with SDP packets are used in signaling between the gateway/the container and the container/the clients; and TCP packets carrying the SOAP traffic between the application and the gateway. The TCP packets constitute the extra load induced by the web service layer. An example of the network load distribution is illustrated in figure 5.9b.

Chapter 6

Conclusions and Future Work

In this chapter, we highlight the contributions of this thesis, and discuss the lessons learned. We also give hints about future research directions.

6.1 Contributions of this thesis

We have presented a case study on the use of Web services for the development of conferencing applications in Internet Telephony, or more precisely in SIP networks. In the study, we have defined comprehensive web service interfaces exposing conferencing capabilities. In addition, a sub-set of these interfaces was implemented as a SIP servlet based gateway. Two audio conferencing applications were built using the implemented interfaces. Performance was evaluated in terms of time delay and network load.

Three main elements make the software architecture of the conferencing server we have built: the gateway, two servlets hosted by a SIP container, and a mixer. Instead of a full-blown SIP container, we used a standalone SIP container providing basic functionality. The gateway and the servlets act as the signaling control unit. The functions implemented by this unit and the mapping of these functions onto SIP were presented. The mixer on the other hand acts as the media control unit. The Java API provided by the mixer was used in the communication between the signaling and the media control units.

The performance measurements were taken for the web service model as well as the raw SIP servlet model. The comparison of these two sets of measurements was used in the calculation of the overhead introduced by the web service layer.

Throughout this work, we have learned a few lessons:

- The first lesson is that the SIP servlet API is a suitable approach for building a web services gateway. In fact, the flexibility offered by the servlet model permitted us to choose the appropriate level of abstraction for our design. In addition, the capabilities offered by this API permit the implementation of advanced signaling functionalities such as conferencing and sub-conferencing. Such functionalities can be used for the development of powerful call control applications. The two applications developed are in fact non trivial and have been fully developed using the call control capabilities implemented.

- The second lesson is that the web service technology promotes easy and fast applications development. In fact, web services hide the implementation details of the call control capabilities exposed. This could allow applications developers who are not necessarily experts in the telecom field to easily integrate call control capabilities to their applications. In addition, using the web services interfaces permits a considerable reduction of the applications foot prints. As observed in the dial-out conferencing application developed, using the web services to incorporate the dial-out call control capabilities requires 70 lines of code while using the traditional SIP servlet approach requires 740 lines of code. This important reduction of the application foot print saves time and effort for application developers.

- The third lesson is that the web service layer does not penalize the system performance in terms of time delays. The average time delay overhead ranges between 43 and 556 ms (3% – 10 % respectively). This delay is barely felt by the application's user. In terms of network load, the extra SOAP traffic represents an overhead ranging between 1.7 and 2.4 Kb (41% - 28% respectively) per functionality. This overhead is considerable however it remains constant with respect to the number of participants, for almost all operations. Other solutions should be investigated to improve the performance of SOAP.

- The fourth lesson is that the current application servers offer a limited support for asynchronous communication. Normally, an application should be able to subscribe

for a number of events (e.g., notify me each time a user joins or leaves) and get notified for each of the events. Presently, only one notification is allowed per subscription. This requires several subscriptions by the application to get notifications for all the events. Improvements should be made to the current tools to offer optimum support for asynchronous communication.

6.2 Future Work

Although the interfaces defined offer many conferencing capabilities, there are some capabilities that are not yet defined. For instance, more floor control features could be added. Examples of these features include expanding and shrinking the scope of a floor, and freezing a floor. In future work, we will look into the definition of these capabilities in order to make the interfaces more comprehensive.

Although the gateway developed implements most of the conferencing functions defined, some functions are not yet implemented. Examples of these functions include floor control related functions and splitting/merging of sub-conferences. In future work, we will look into the implementation and the mapping of these functions onto SIP.

Although the mixer used offers support for both audio and video streams, the interfaces defined uses the voice capability only. In the future, we would like to also use the video capability to allow video conferences. In addition, the support of several codecs should be investigated.

One of the limitations of the application server used is the limited support to asynchronous communication. Presently, several calls to the getConfUpdate () function are to be made by the application to get all the conference updates. A more efficient way to support asynchronous communication is among our interests.

Although the extra SOAP traffic introduced by the web service layer does not significantly affect the time delay, it generates a considerable network load overhead. We would like to find a solution in order to decrease the weight of the SOAP messages used.

Many applications can be developed in Internet telephony using call control capabilities. However, some applications may need other capabilities such as presence and location. Extensions to SIP have been defined to support presence. The composition of the call control services with other types of services such as presence in order to provide building blocks for applications is an also among our interests.

REFERENCES

[1] N. Alameh, "Chaining Geographical Information Web Services," *IEEE Internet Computing Magazine*, vol. 7, no. 5, pp. 22-29, September 2003

[2] BEA Weblogic tutorial on designing asynchronous interfaces at http://edocs.bea.com/workshop/docs81/doc/en/core/index.html

[3] The CGI specification at http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

[4] H. Dommel and J. Aceves, "Floor Control for Multimedia Conferencing and Collaboration," *ACM Multimedia Systems Magazine*, vol. 5, no. 1, pp. 23-38, 1997

[5] S. Donovan, "The SIP INFO Method," RFC 2976, IETF, October 2000

[6] ECMA-348 call control specifications at http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-348.pdf.

[7] ESTI Official web site at http://www.esti.org/

[8] Ethereal web site at http://www.ethereal.com/

[9] R. Glitho, R. Hamadi and R. Huie, "An Architectural Framework for Using Java Servlets in a SIP environment," *ICN 2001*, Colmar, France, July 2001

[10] R. Glitho, F. Khendek, and A. De Marco, "Creating Value Added Service in Internet Telephony: An Overview and a Case Study on a High-Level Service Creation Environment," *IEEE Transactions on systems, man and cybernetics*, vol. 33, no. 4, pp. 446-457, November 2003

[11] R. Glitho and K. Sylla, "Developing Applications for Internet Telephony: A Case Study on the Use of Parlay Call Control APIs in SIP Networks," accepted for publication by IEEE Network

[12] N. Greene, M. A. Ramalho, and B. Rosen, "Media Gateway Control Protocol Architecture and Requirements," RFC 2805, IETF, April 2000

[13] M. Handley *et al.*, "SIP: Session Initiation Protocol," RFC 2543, IETF, March 1999

[14] M. Handley and V. Jacobson, "SDP: Session Description Protocol," RFC 2327, IETF, April 1998

[15] S. Hawwa, "Audio Mixing for Centralized Conferences in a SIP Environment," *IEEE International Conference on Multimedia*, August 2002

[16] ITU-T Recommendation H.323, "Packet based multimedia communications systems," Geneva, 2002

[17] JAIN JCC/JCAT, JSR 21 at http://www.jcp.org/jsr/detail/21.jsp

[18] Java Servlet Specification Version 2.3, JSR 00 053 at http://jcp.org/aboutJava/communityprocess/final/jsr053/

[19] JCP Java SIP Servlet API, JSR 116 at http://jcp.org/aboutJava/Communityprocess/review/jsr116/

[20] A. Johnston, *SIP: Understanding the Session Initiation Protocol*, Artech House Inc, Norwood, Massachusetts, 2001

[21] J. Lennox, J. Rosenberg, and H. Schulzrinne, "Common Gateway Interface for SIP," RFC 3050, IETF, January 2001

[22] L. Lennox and H. Schulzrinne, "Call Processing Language Framework and Requirements," RFC 2824, IETF, May 2000

[23] J. Lennox and H. Schulzrinne, "transporting User Control Information in SIP REGISTER Payloads," Internet draft, IETF, March 1999, work in progress

[24] H. Liu and P. Mouchtaris, "Voice over IP Signaling, H.323 and Beyond," *IEEE Communications Magazine*, vol. 38, no.10, pp. 142.148, October 2000
[25] R. Nagappan, R. Skoszylas, and R. Sriganesh, *Developing Java Web Services*, Wiley Pulishing Inc, Indianapolis, Indiana, 2003
[26] OASIS standards consortium web site for UDDI at http://www.uddi.org/
[27] The OMA Web Services Enabler core specification v1.1 (16-02-2004) at http://member.openmobilealliance.org/ftp/public_documents/mws/2003/
[28] The Organization for the Advancement of Structured Information Standards official website at http://www.oasis-open.org/home/index.php
[29] Parlay 4.1 specifications at http://www.parlay.org/specs/index.asp
[30] The Parlay X specifications at http://www.parlay.org/specs/index.asp
[31] A. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification," RFC 3265, IETF, June 2002
[32] J. Rosenberg et al, "SIP: Session Initiation Protocol," RFC 3261, IETF, June 2002 (Obsoletes RFC 2543)
[33] J. Rosenberg, J. Lennox, and H. Schulzrinne, "Programming Internet Telephony Services," *IEEE Internet Computing Magazine*, vol. 3, no. 3, pp. 63-72, May-June 1999
[34] J. Rosenberg and H. Schulzrinne, "Models for Multi Party Conferencing in SIP," Internet Draft, IETF, May 2001, work in progress
[35] J. Roy and A. Ramanujan, "Understanding web services," *IEEE IT professional Magazine*, vol. 3, no. 6, pp. 69-73, November 2001
[36] H. Schulzrinne and K. Arabshian, "Providing Emergency Services in Internet Telephony," *IEEE Internet Computing Magazine*, vol. 6, no. 3, pp. 39-47, May-June 2002
[37] H. Schulzrinne and J. Rosenberg, "Signaling for Internet Telephony," *Sixth International Conference on Network Protocols*, pp. 298-307, October 1998
[38] H. Schulzrinne and J. Rosenberg, "The Session Initiation Protocol: Internet Centric Signaling," *IEEE Communications Magazine*, vol. 14, no. 4, pp. 134-141, October 2000
[39] K. Singh, Gautam Nair, and H. Schulzrinne, "Centralized Conferencing Using SIP," *In Internet telephony workshop 2001*, New York, April 2001
[40] H. Sinnreich and A. Johnston, *Internet Communications Using SIP*, John Wiley and Sons Inc, New York, 2001
[41] R. Sparks, "The Session Initiation Protocol (SIP) Refer Method," RFC 3515, IETF, April 2003
[42] Third Generation Partnership Project (3GPP) at http://www.3gpp.org/
[43] T. Thompson, R. Weil, and M. Wood, "CPXe: Web Services for Internet Imaging," *IEEE Computer magazine*, vol. 36, no. 10, pp. 54-62, October 2003
[44] W3C web site for SOAP at http://www.w3c.org/TR/SOAP/
[45] W3C working draft of the Web services architecture specification at http://www.w3.org/TR/2003/WD-ws-arch-20030808/
[46] W3C web site for WSDL at http://www.w3c.org/TR/wsdl/
[47] W3C web site for XML at http://www.w3c.org/XML/
[48] The Web Services Interoperability organization official website at http://www.ws-i.org/

[49] The World Wide Consortium official website at http://www.w3.org/
[50] WSDL document for ECMA's call control interfaces at http://www.ecma-international.org/standards/ecma-348/csta-wsdl/