IT 08 017

Examensarbete 30 hp
Maj 2008

Web Call SDK

SIP Stack in Web Container Component

Yuening Chen

Institutionen för informationsteknologi
Department of Information Technology

Abstract

Web Call SDK: SIP Stack in Web Container Component

Yuening Chen

The Internet and the Web are the dominant communication methods nowadays. There is also a trend in using web technology to replace or supplant the more traditional communication methods, for example using the internet to make Voice over IP telephone calls.

This thesis proposes an efficient solution to integrate VoIP/SIP call control functionality into Web containers. It presents the communication convergence of Web technology, VoIP networks, and PSTN circuit switched (CS) networks.

The Web Call Software Development Kit (SDK) is the implementation of this solution. It consists of three sub-projects: a SIP Call Control Component, a web application, and a Java ME Client Application. Using the SDK it is possible to easily implement SIP call functionality in Java EE web containers and other Java applications.

Handledare: Peter Yeung
Ämnesgranskare: Olle Eriksson
Examinator: Anders Jansson
IT 08 017
Sponsor: Ericsson AB

Tryckt av: Reprocentralen ITC

Contents

1 Introduction

1.1 Background

The Internet and the Web are the dominant communication methods nowadays. There is also a trend in using web technology to replace or supplant the more traditional communication methods, for example using the internet to make Voice over IP telephone calls.

IP telephone (VoIP) technology enables the communication between Internet users and endpoints in PSTN circuit switched (CS) networks. As we know, the Session Initiation Protocol (SIP) is widely adopted as a signaling protocol for VoIP communications. Because of its simplicity, power and extensibility, it has also been selected by the Third Generation Partnership Project (3GPP) as the main session control protocol of the core network of the IP Multimedia Subsystem (IMS).

Based on IMS/SIP network technology, the Web Call SDK integrates SIP call control functionality into web containers. This presents an efficient way of implementing the communication convergence of Web, IMS/SIP network, and CS networks. It does not require the installation of client-side browser plug-ins or special client software as most other VoIP services require.

Compared to other customer-oriented VoIP products, the Web Call SDK offers solutions for developers. The SDK makes it easy to integrate SIP call functionality in web containers using standard Java EE components and thus allows easy and innovative integration of web, IMS/SIP, and mobile communication technology.

1.2 Goal

The objective of this master thesis consists of three aspects:

- Theoretical research
- Implementation
- Functionality improvement

The theoretical research involves the comparison of existing SIP control techniques, an evaluation of similar commercial products, a marketing analysis, and a implementation proposal. This pre-study presents a systematic analysis and provides motivation for the implementation proposal.

The Web Call SDK is the implementation of the proposal. The main SIP call functionality is called the Call Control Component which serves to establish a session between participants. The Call Control Component is integrated into a web application as several different Java EE components. The web application provides services for web users, WAP users, and Java ME users.

Functionality improvement is done in coordination with the implementation and focuses on improving the performance to meet the design goal of efficiency, compatibility, and extendibility.

The main innovative contribution of this thesis is to research, design and develop a SIP third party call control component and the related applications. This presents a new, precursory, and effective solution in web communication technology.

1.3 Overview

The Web Call SDK consists of the Call Control Component, a web application, a custom tag, and a Java ME Client application. The purpose of SDK is to provide a high level API to implement SIP call control, and to demonstrate how to integrate the SIP call component in a web container using Java Beans and web services. It also includes a Java ME client example to access the web service included in the web application for making phone-to-phone SIP calls. This SDK can help Java application developers to understand how the Call Control Component works and how Java EE and Java ME applications can be developed using it. It can also be used for connecting to a SIP based network such as the IMS core network. Below is a diagram indicating how all the components in the SDK interact.

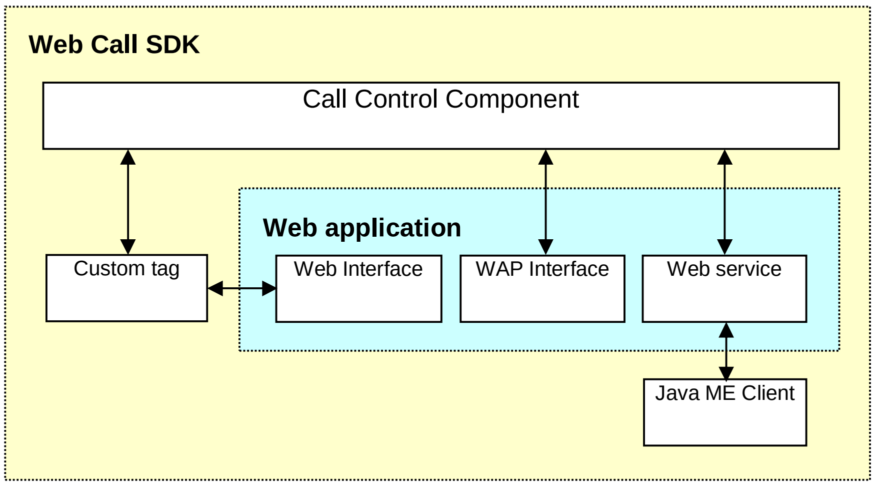


Figure 1   Web Call SDK structure

Call Control Component

The Call Control Component provides a high-level API for developers to implement SIP phone-to-phone call functionality. The component is built using the open source MJSIP library. Its main functions include establishing SIP sessions between two endpoints, controlling the session states, and managing the audio proxy. The Call Control Component is a SIP call-controlling component, connecting SIP endpoints through the SIP networks selected by the users.

Web Application

The web application contains a set of examples for developers, showing how to integrate the Call Control Component using Java Beans, a custom JSP tag, and a web service into a Java EE web container. The web application uses the Front Controller architecture and provides different views for both Web and WAP browsers. It provides an easy to use, efficient, and secure web application for both web and mobile users.

Java ME Client

The Java ME client is related to the web application. It is a client-side Java ME application, which uses SOAP to access the phone-to-phone web service integrated in the web application.

1.4 Network overview

Figure 2 shows an overview of the network architecture for a service provider using the web application. The application runs on a Java EE web container and resides outside the operator's mobile network.

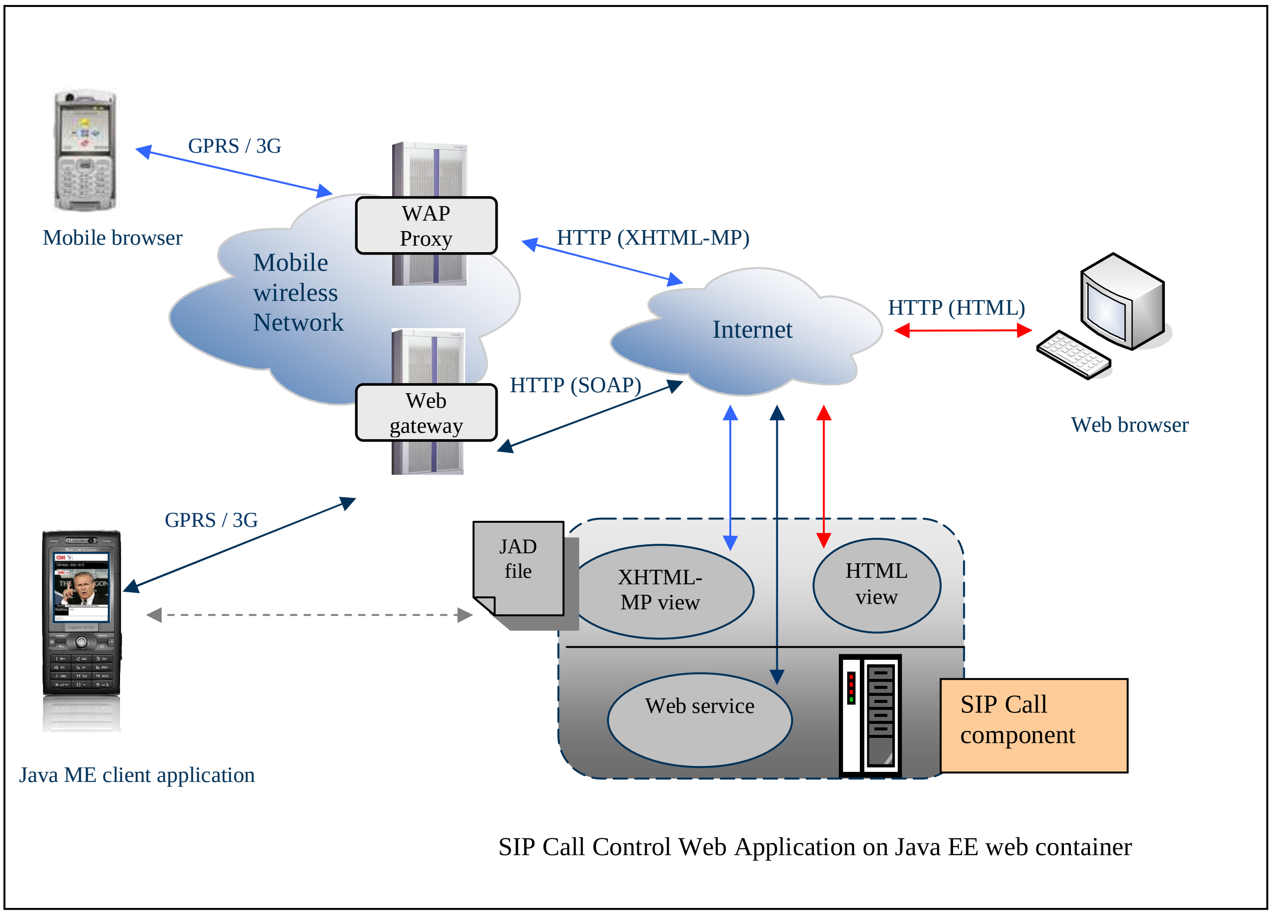


Figure 2 Network overview

At the front-end of the web application, there are three types of views or ways of accessing the web application:

- Web users can use a web browser to visit the web application on the internet through their ISP. The web application provides a HTML version for web users.

- Mobile users can use a WAP browser to visit the web application via their mobile operator’s network. The web application provides different XHTML-MP interface versions for mobile user having either a big or a small screen.

- More advanced mobile user, whose mobile supports the JSR 172 specification, can download and install the Java ME client application, and access the web services via their mobile operator’s network.

To use the third option, users access the WAP version and choose to download the client application. The web application then dynamically generates the Java ME client application.

Basic Use cases

The Web Call SDK has implemented the following basic use cases:

- SIP phone-to-phone call functionality: a desktop application for setting up a SIP session between phones using a user provided SIP account.

- Web phone-to-phone call functionality: a web application for setting up a SIP session between phones using a user provided SIP account.

- Click-to-Call business service: a web application for setting up a session between a predefined phone and a user's phone, using a predetermined SIP account.

- Java ME Client phone-to-phone call functionality: a Java ME client application for setting up a SIP session between phones using a user provided SIP account.

These use cases will be discussed in more detail in the upcoming chapters.

1.5 Scope

The programming environment for this thesis is based on Java technology, which is one of the main standards for web and mobile internet services in the service layer. All the descriptions are based on Java SE/EE/ME technology.

1.6 Related work

There are a lot of VoIP operators providing VoIP services, but almost all the services are commercial closed solutions. So the available information is very limited. Skype is very popular in the VoIP field [1]; SkypeIn and SkypeOut are using SIP as the signaling protocol, but most of the internal techniques are private. It also requires installation of client-side software, which a common property of most VoIP operators.

There are very few products offering phone-to-phone services; the common use cases are PC-to-Phone and PC-to-PC. Skype published their mobile version recently, consisting of three variants: Skypephone, Smartphone/Windows phone, WIFI phones. Implementation details are not available for any of these variants.

VoipStunt and related products (FreeCall, Nonoh) opened their services to SIP customers. Customers can configure their SIP devices or client software to access the SIP servers. VoipStunt also provides phone-to-phone services in their client software, and trial calls on their web site. [2]

Compared to the products in the current SIP market, the Web Call SDK is open source. Its Call Control Component is an open library, and conforms to the SIP standard. It can be configured to access SIP services with the user’s own SIP account. The innovation of the Web Call SDK is that it not only provides an open source solution, but also includes several other functions: SIP call functions for web users, WAP users, as well as mobile application users.

1.7 Report outline

This thesis report starts with a background introduction to IMS/SIP networks, VoIP, Java EE and Java ME technologies in Chapter 2. In Chapter 3, we introduce a comparison of SIP session control techniques, evaluate their performance and list their limitations. Chapter 3 concludes with an implementation proposal for the Web Call SDK.

The report then describes three sub-projects each in a separate chapter; 4 for the Call Control Component, 5 for the web application, and 6 for the Java ME client. Each chapter contains the design goal, problem description, a description of the proposed solution, design features, and an evaluation of the results or a comparison to other implementations.

At the end of the report, an integral conclusion is given in Chapter 7, as well as a description of future work in Chapter 8.

1.8 Appendix

The documentation for the Web Call SDK is appended to this thesis. It contains documentation for the three projects: the Call Control Component, the web application, and the Java ME Client, as well as relevant documentation such as JavaDoc and a developer’s guide. The developer’s guide describes the design and usage of the Web Call SDK in more detail.

2 Background Concepts

Before analyzing and describing the solution, this chapter gives a brief introduction to the technologies used in this project: IMS/SIP, VoIP, Java EE, and Java ME.

2.1 IMS/SIP network

The IP Multimedia Subsystem (IMS) is a generic architecture for implementing IP based telephony and multimedia services. It was originally designed by the 3rd Generation Partnership Project (3GPP), a wireless standards body. Its standardized reference architecture enables the convergence of voice, video, data and mobile network technology over an IP-based infrastructure.

IMS uses the Session Initiation Protocol (SIP) as signaling protocol in call/session control, which a major component in the UMTS core network. SIP is a control signaling protocol at the application-layer for creating, modifying, and terminating sessions with one or more participants. It can be used in a traditional two-way audio session, or as a video and multimedia conference session. The SIP specification is in IETF RFC 3261 [3].

SIP works with several related protocol to establish a concrete media session. The most common protocols are the Session Description Protocol (SDP) and the Real-time Transport Protocol (RTP).

SDP describes the media parameters of the multimedia session, for example, the IP ports for media transfer, the encoding formats, etc. It is carried by SIP in the session initiation process for session invitation or session announcement. SDP is defined in IETF RFC 2327.

RTP is the actual protocol which manages the audio/video packets transport in a SIP session. After a successful session initiation, one or more RTP streams are set up according to the SDP negotiation phase. RTP is specified in IETF RFC 1889.

2.2 VoIP technology

2.2.1 VoIP introduction

Generally speaking, Voice over Internet Protocol (VoIP) is the technology that enables users to carry voice traffic — for example, telephone calls, MMS, and video — over an IP-based network. It is commonly called IP telephony.

VoIP technologies are in wide use these days. Various specifications and functions are implemented by VoIP providers. For example, Skype, VoIPstunt, FreeCall, etc. As we know, in most cases, the IP telephone calls cost less than traditional Circuit-Switch calls. When the users are in the same IP-based network, the cost to carry to media, including audio, video, and messages are usually free. However, the provider usually charges a small fee for a connection to a network other than their own IP-based network. Take VoipStunt for example, to make a call from an active end-user (VoIP phone) to a Tele2Comviq user, the cost is currently 0.10 Euro/min. In this case, the VoIP calls connect to the public switched telephone network via a VoIP-PSTN gateway, which is operated by the providers. Figure 3 shows the network of a typical VoIP-to-PSTN call.

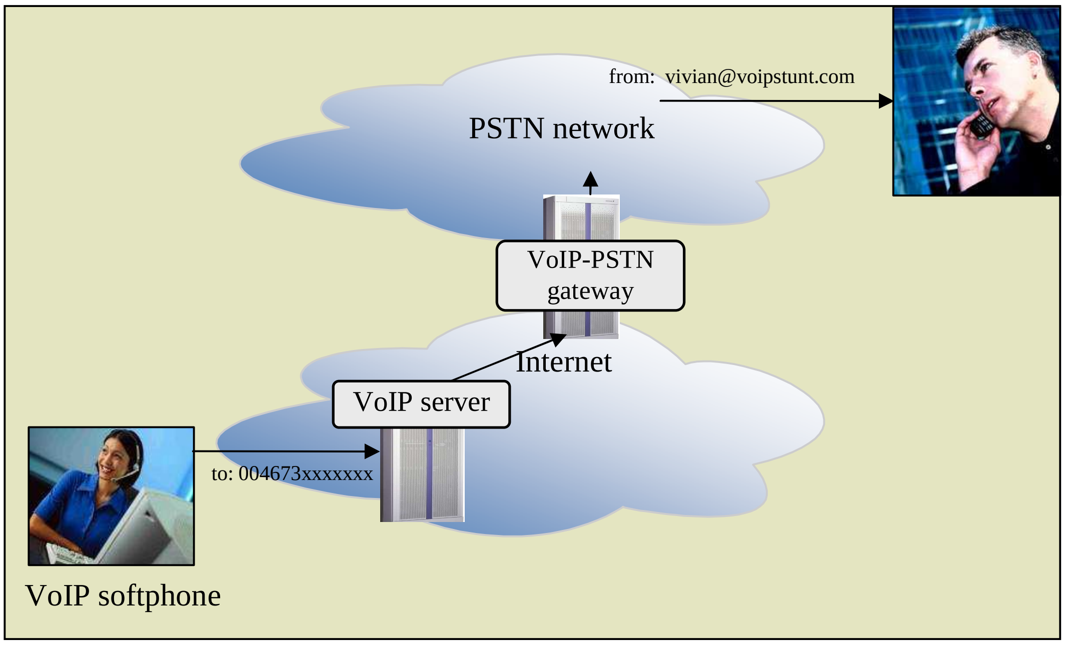


Figure 3   VoIP-PSTN network

2.2.2 SIP technology in VoIP

There are several VoIP signaling protocols at present, such as SIP, H.323, and MGCP (Media Gateway Control Protocol). The following table below is re-formatted from “Voice Over IP and VOIP Protocols” [4]. It is a general comparison between these often used signaling protocols.

	SIP	H.323	MGCP/H.248/Megaco
Standards body	IETF	ITU-T	MGCP/Megaco – IETF H.248 – ITU-T
Architecture	Distributed, Peer-to-Peer	Distributed	Centralized
Call Control	Proxy/Redirect Server	Gatekeeper	Call agent/Media Control Gateway / Softswitch
Endpoints	User agent	Gateway, terminal	Media Gateway
Signaling Transport	TCP/UDP	TCP/UDP	MGCP – UDP H.248/Megaco – TCP/UDP
Multimedia	Yes	Yes	Yes
DTMF-relay transport	RTP – Real Time Transport Protocol	RTP – Real Time Transport Protocol	RTP – Real Time Transport Protocol
Fax-relay transport	T.38	T.38	T.38
Supplemental services	By endpoints or call control	By endpoints or call control	By call agent

Although many other VoIP signaling protocols exist, SIP is considered as scalable, simple, and easy to implement. The main advantage of SIP is that it supports both IP and traditional telephone sessions. It can be used in a distributed system as well as in a peer-to-peer system. Recently, more and more VoIP providers prefer the SIP protocol stack because it is easier to leverage other functions on SIP and implement the interconnection between IP-based network and traditional Circuit-Switch networks. Moreover, because SIP's structure is similar to HTTP, voice based applications can also be seamlessly integrated with web services. SIP has been selected as the standard for call control and signaling in IMS networks. The following figure shows the basic architecture of a SIP-based VoIP network. The message flows in this figure are examples, for more detail, see the flow charts in section 4.3 - Session Control Call Flows.

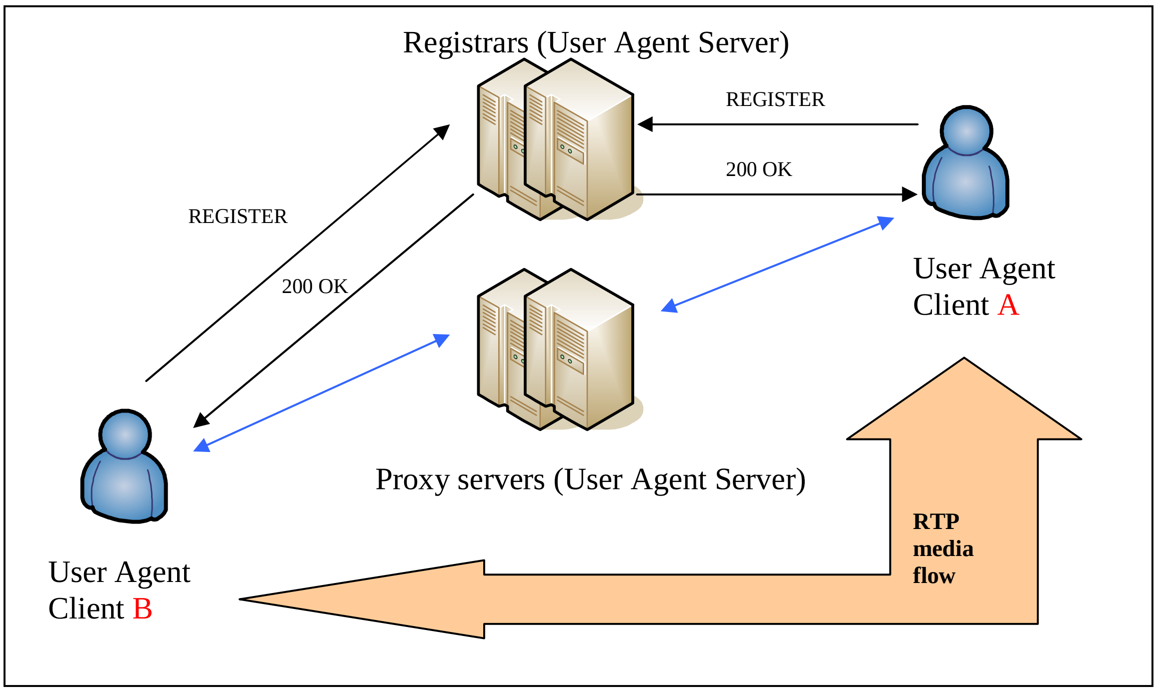


Figure 4 SIP-based VoIP network

2.3 Java EE technology

2.3.1 Java EE introduction

The Java Platform Enterprise Edition (Java EE) builds on the Java Platform Standard Edition (Java SE) and is one of the leading industry standards for implementing enterprise-class service-oriented architectures (SOA) and next-generation web applications. [5]

N-tier architecture

Java EE provides a unified specification to build a distributed enterprise application. The N-tier architecture is an important feature of an efficient Java EE enterprise application. It contains a user interface layer (which could be a web browser), presentation logic, business logic, infrastructure services, and a database layer. Figure 5 shows the architecture of a N-tier web application.

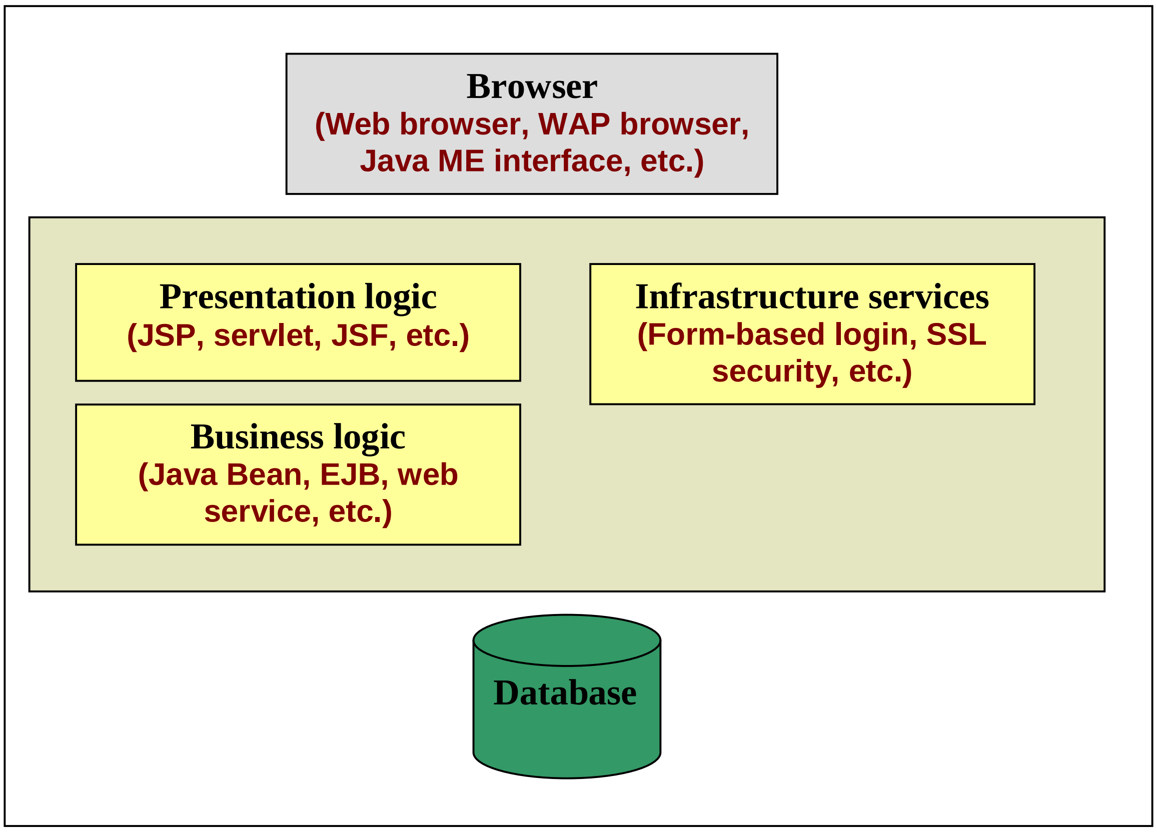

Browser
(Web browser, WAP browser, Java ME interface, etc.)
Presentation logic
(JSP, servlet, JSF, etc.)
Infrastructure services
(Form-based login, SSL security, etc.)
Business logic
(Java Bean, EJB, web service, etc.)
Database

Figure 5 N-tier architecture of the web application

2.3.2 MVC pattern

The N-Tier architecture is based on the Model-View-Controller (MVC) design pattern. It allows for application to be loosely coupled, extensible, while sharing a common data source. The MVC model consists of:

- Model: Represents the shared data and provides methods to operate on the data.
- View: The display of the Model, for example using JSP pages.
- Controller: Handles events that affect the model or view(s).

The Web Call SDK uses the MVC pattern as well. The advantage of using the MVC model is that the layers are separately constructed, so modification or extension is independent and convenient. The components are loosely coupled which improves re-usability.

2.3.3 Java EE components

There are various components in Java EE applications, for example JSP pages, Servlets, Java Beans, custom tags, etc. The following introduction mainly focuses on the components that are used in the Web Call SDK.

Custom tag

Developers can define a custom tag to hide implementation complexity inside a JSP tag. The execution of custom tags is the same as the JSP tags in JSTL (JSP Standard Tag Library). When the tag is called at runtime its tag handler methods are called, in turn executing the custom code.

The main advantage of using custom tags is that it hides implementation details from the view layer. Web designers can easily read the TLD specification and use the custom tag without knowing the implementation details.

Java Beans

A Java class is said to be a Java Bean when its interface meets certain requirements. This makes it a reusable software component, which — among others — can be visually manipulated in builder tools. The following list is excerpted from [6], and lists the requirements for a Java class to be used as a Java Bean:

- Must be serializable. Because they might be saved into an object database.
- Must have a parameter-less constructor.
- Name convention of the instance variables and other resources.
- Getter and setters. Methods are used to retrieve and modify resources.

Web service

Web services are an essential technology in Service-Oriented Architecture (SOA). Web services often use XML for data exchange, but also might take other data formats such as JSON, CSV, etc. [7] Web services are commonly implemented in the form of Web API's, and can be accessed through a variety of protocols, such as SOAP, REST (HTTP), or some other form of RPC. The execution is performed on a remote server hosting the requested services.

2.4 Java ME technology

The Java Platform Micro Edition (Java ME) provides a robust and flexible environment for applications running on mobile and other embedded devices — mobile phones, personal digital assistants (PDAs), TV set-top boxes, and printers. [8]

Generally speaking, Java ME is a curtailed version of Java SE. It targets applications running on devices with limited processing power and storage capabilities, or other limited system resources. Mobile applications are the common use of Java ME technology

Java ME Web service technology

The J2ME Web Services APIs (WSA) extends the web services platform to include J2ME. The J2ME Web Services APIs enable J2ME devices to be web service clients, and provides a programming model that is consistent with the standard web services platform. [9]

WSA implements JSR 172 – Java ME Web Service Specification. A mobile phone should support this specification to use the web service function. The Sun Java Wireless Toolkit provides tools to generate web service stubs from WSDL (Web Service Description Language) and a mobile simulator for developers.

2.5 Terminology

CS: Circuit Switching. The act of establishing a fixed bandwidth circuit between nodes before communication commences.

IMS: IP Multimedia Subsystem. An architecture for delivering internet protocol multimedia to hand-held devices.

JAD: Java Application Descriptor. A file describing meta-data for MIDlets.

MIDlet: A Java application for mobile devices that support the Java ME VM.

PBX: Private Branch eXchange. An internal telephone network commonly used in businesses and other organizations.

PCM U/A: A common media format supported by the RTP protocol.

SIP: Session Initiation Protocol. A protocol for creating, modifying, and terminating sessions with one or more participants.

SDP: Session Description Protocol. A protocol for describing streaming media sessions, such as audio and video formats.

PSTN: Public Switched Telephone Network. A reference to the public telephone network world-wide including both land-lines and mobiles.

RTP: Real-time Transport Protocol. A carrier for audio/video communications.

UAC: User Agent Client. An application that behaves like a client in the SIP protocol.

UAS: User Agent Server. An application that behaves like a server in the SIP protocol.

WAP: Wireless Application Protocol. A standard for wireless communication, usually to access the Internet from hand-held devices.

XHTML-MP: XHTML Mobile Profile. A version of the XHTML standard targeted to mobile and hand-held devices.

3 SIP technique comparison

The SIP call control function required in this project is a call-controlling component for SIP networks. The main function such a component is to establish a conversation between two end-points and manage the session. There are some existing SIP techniques which also implement the SIP-session controlling function. This chapter compares three common techniques: third-party-call-control (3PCC), Back-to-Back User Agent (B2BUA), and Call transfer. It evaluates these solutions and explains why the Call Control component is used in this project instead of existing techniques.

3.1 Third Party Call Control

3.1.1 Introduction

The 3PCC technique is defined in “RFC 3725 - Best Current Practices for Third Party Call Control (3PCC) in the Session Initiation Protocol (SIP)”. It is an extension of the standard SIP specification.

The main entity in 3PCC is a controller, which initializes the session between two or more other parties and manages the communication control. 3PCC provides four possible flow controls that can be utilized in order to provide basic operation.

The following is the fourth flow chart from RFC3725, which is described to have many benefits. [10]

```
A                 Controller                  B
|(1) INVITE offer1     |                      |
|no media              |                      |
|<---------------------|                      |
|(2) 200 answer1       |                      |
|no media              |                      |
|--------------------->|                      |
|(3) ACK               |                      |
|<---------------------|                      |
|                      |(4) INVITE no SDP     |
|                      |--------------------->|
|                      |(5) 200 OK offer2     |
|                      |<---------------------|
|(6) INVITE offer2’    |                      |
|<---------------------|                      |
|(7) 200 answer2’      |                      |
|--------------------->|                      |
|                      |(8) ACK answer2       |
|                      |--------------------->|
|(9) ACK               |                      |
|<---------------------|                      |
|(10) RTP              |                      |
|.............................................|
```

Figure 6   3PCC flow chart IV

During the session establishment, the controller sends two INVITE requests to client A, one without media information in SDP, and the other one with the SDP offer it received from client B (by extracting it from the B’s response to the INVITE request).

After A answers the INVITE request with 200 OK for the second time, the controller sends the ACK to both sides to confirm the session.

When the session is established, the RTP media transfers is carried out peer-to-peer between A and B. The controller will be separated from the media transfer, but will still take control of the SIP session, for example, closing down the session.

3.1.2 Evaluation of 3PCC

The original design of the Call Control Component followed this control flow as well, because it seemed like the best match to the project requirements.

Desired functionality

As mentioned earlier, the functional requirements of the Call Control Component are:

- Establish a session between two end-points.

- Control the session during the conversation, for example, session termination.
- Two end-points should be able to have continuous bi-channel media communication.

A solution based on 3PCC can meet al the requirements above.

Efficiency

The 3PCC controller negotiates the media information for three-rounds (A -> B -> A), as shown in Figure 6. Once the session has been set up, the controller will ‘leave’ the real media transfer, and acts as a supervisor, controlling the session modification and termination. The RTP media will be exchanged peer-to-peer between two end-points.

In this way, the controller does not participate in the media transferring, so the RTP communication is very efficient compared to other proxy system in that it does not involve a third party for media transfers.

3.1.3 Implementation issues

Compatibility problems

The 3PCC prototype was implemented at the beginning of the project, and it was tested against various commercial and experimental SIP servers.

However, the compatibility of this technique is not optimal at this time. Because it is an extension to the SIP standard, and it is a relatively new technology, most products in the SIP network do not yet support this feature. These products include not only the SIP server but also the VoIP-PSTN gateways and other client devices. Unfortunately, those servers/gateways are either not open (for example, most SIP providers), or have no available documentation. This makes it even more difficult to research or implement.

The 3PCC prototype has been successfully tested in a peer-to-peer model with two MjSipUA clients [11]. But due to limited support from actual SIP server and gateways, this solution, unfortunately, can not be put into practice at the moment.

3PCC is a valuable technique which, hopefully, can be used in a few years when the SIP networks support this feature.

3.2 B2BUA

Back-to-Back User Agent (B2BUA) is usually compared to a SIP proxy. Signaling gateways and Asterisk PBX [12] are usually mentioned as examples of B2BUA. Figure 7 is a simple use-case of B2BUA. It shows the relation between the B2BUA and two end-points. It can also be used in handling multiple participants, but this is not shown in this example.

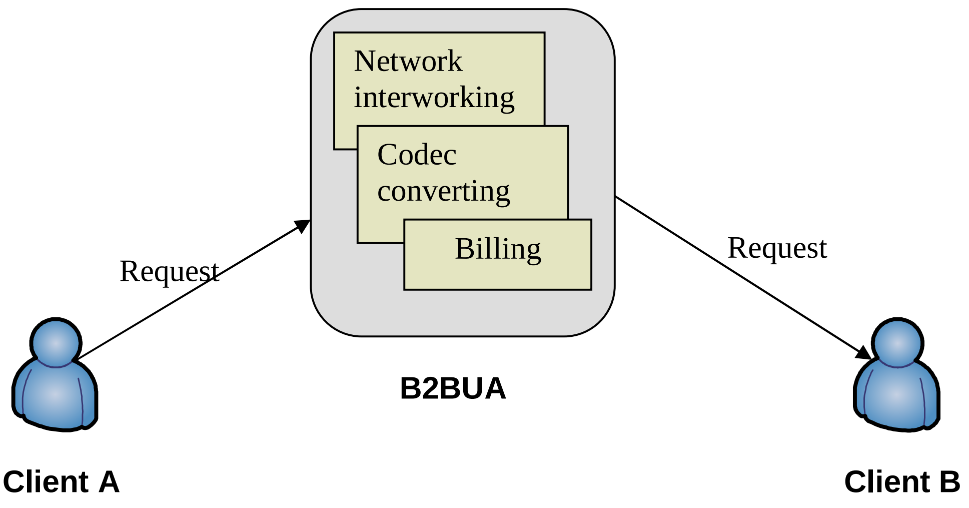


Figure 7 B2BUA use case

Although B2BUA is called User Agent, it is acting as a server on one end and a client on the other. If the B2BUA is a call control component, it receives an incoming call from the originating SIP endpoint (Client A) and then initiates another session to the other endpoint (Client B).

The main difference between a B2BUA and a SIP proxy is that the B2BUA can add features such as billing, network interworking and codec converting.

Functionality comparison

As mentioned earlier, the main functional requirement of the SIP controller is to initialize a session between two end-points. The SIP controller should be the originating entity.

However, as mentioned in the previous example, the B2BUA receives the incoming call from one side and then initiate another call to the other side. It is activated by at least one end-point. This feature does not satisfy the initialization functionality for a Call Control Component.

Moreover, if we want to establish a session between two normal mobile phones, then the charging system will be very complicated. Usually, the mobile provider will charge the mobile phone is it initiates the call. But in this project, we want to make phone calls using the SIP account from regular SIP providers.

3.3 Call transfer

Call transfer has similar functionalities as B2BUA, and is thus not a proper solution either. Additionally, call transfer uses extended SIP messages such as the REFER method, which is not commonly supported by SIP servers and gateways, so it might cause compatibility problems.

3.4 Call Control Component

The Call Control Component is an innovative solution for implementing a third-party call controller. Compared to 3PCC, it has high compatibility, and it functions as a RTP proxy.

Solution for current SIP networks

As described in section 3.1, 3PCC is characterized as having the desired functionality and good efficiency, but it has serious compatibility problems considering the current SIP networks.

To solve the compatibility problems, and keep the functionality and efficiency, the Call Control Component proposes a new solution, using basic SIP messages, which adheres to the SIP specification in RFC3725. And it also provides RTP proxy functions for the end-points.

Solution summery

The solution is detailed in Chapter 4. What follows is a brief introduction to the Call Control Component. The following list includes the important design choices in the solution.

- The controller initializes two separate sessions to both clients.
  - The controller defines the media information in its INVITE message.
  - The two participants compare the media information with their own media types, reach an agreement, and send the response back to the controller.
- The controller maintains the sessions between the two clients.
  - After both sessions are established, the controller maintains the session status.
  - During the conversation, the controller will not initialize any request until receiving a signaling message from the clients.
- The controller proxies the signaling in the termination process.
  - When the controller receives a signaling message from one client such as BYE, CANCEL, 4xx (Error, busy, no response, etc), it sends the corresponding signal to the other client.
- The controller acts as a RTP proxy in the media transfer.
  - While one end-point (Client A) is waiting for the other (Client B) to answer the phone, the controller sends a ring back tone to Client A, indicating it is trying to reach Client B.

- After both clients answer the phone, the controller receives their RTP packets and proxies them to the other client.
- If two sessions have different RTP formats, the controller converts the media formats of the received packets and forwards them to the other client.

The solution implementation and design features are described in more detail in the next chapter.

4 Call Control Component Solution Description

This chapter gives the solution description and design features of the Call Control Component. The design issues, result evaluation and future work are also included.

4.1 Design goal

The role of the Call Control Component is to function as a session-control module. It provides a high-level API for developers to implement SIP session management between endpoints as well as serving as a media proxy. This call controlling entity can be used for operator services, where an operator creates a call that connects two participants or more (conference calls) together.

The design goal of the Call Control Component is that it should be:

- **Light-weight.** The component should be centralized and light-weight, because it is going to be integrated in different applications, which are server-side or service-oriented. So the cost of execution should be small.
- **Extendable.** The component should be easy to extend. For example, the RTP proxy can be extended with other sub systems to handle video and text message communication.
- **Efficient.** The component is a third-party-call controller between two endpoints. The session between users should be efficient and the latency in the communication should be small enough to maintain the call quality.
- **Transparent.** The component should be transparent during a conversation. That is, it should not in any way influence the conversation.
- **Reusable.** The models in the component should be loosely coupled, so they can be reused in other functions.

- **Robust.** The component should be able to recover from certain failures such as failing connections, and out of service notices from SIP servers.

4.2 Problems

The original design was based on Third Party Call Control (3PCC). 3PCC allows one entity (called the controller) to set up and manage a communication relationship between two or more other parties. When the session has been setup, the media communications are transferred peer-to-peer between the parties. Thus, the controller is very functional and light-weight by using the 3PCC specification.

However, as mentioned in section 3.1, the use of 3PCC is limited when aspects of the call utilize SIP extensions or optional features of SIP. The extensions are specified in “RFC 3725 - Best Current Practices for Third Party Call Control (3PCC) in the Session Initiation Protocol (SIP)”, and “RFC 3515 - The Session Initiation Protocol (SIP) Refer Method”.

The 3PCC controller prototype was implemented at the beginning of the project; it was functional when testing with two endpoints in a peer-to-peer model. It however encountered problems when put into a real SIP network environment. The reason is that most SIP servers or SIP-PSTN gateways don’t support the extended features of SIP, for example, some servers consider INVITE messages without SDP as a bad request.

4.3 Solution proposal

To manage sessions and implement media transport, the Call Control Component has to manually setup and manage sessions, and meanwhile act as a media communication proxy. The following design points are required for the implementation of said functionality:

- Initialize session requests to clients
- Response to the received SIP messages from the clients

- Coordinate between sessions, e.g. phone-to-phone, conference
- Media packets proxy
- Error handling
- Extendibility
- Flexibility

4.4 Design architecture

The Call Control Component acts as a SIP User Agent Client (UAC) in a SIP network. It uses the MjSip v1.6 toolkit as the lower level SIP stack, and implements phone-to-phone session control and RTP proxies on top of it. The Call Control Component provides a high level API, so it is very convenient for developers to use and extend. Figure 8 shows the design architecture of the Call Control Component.

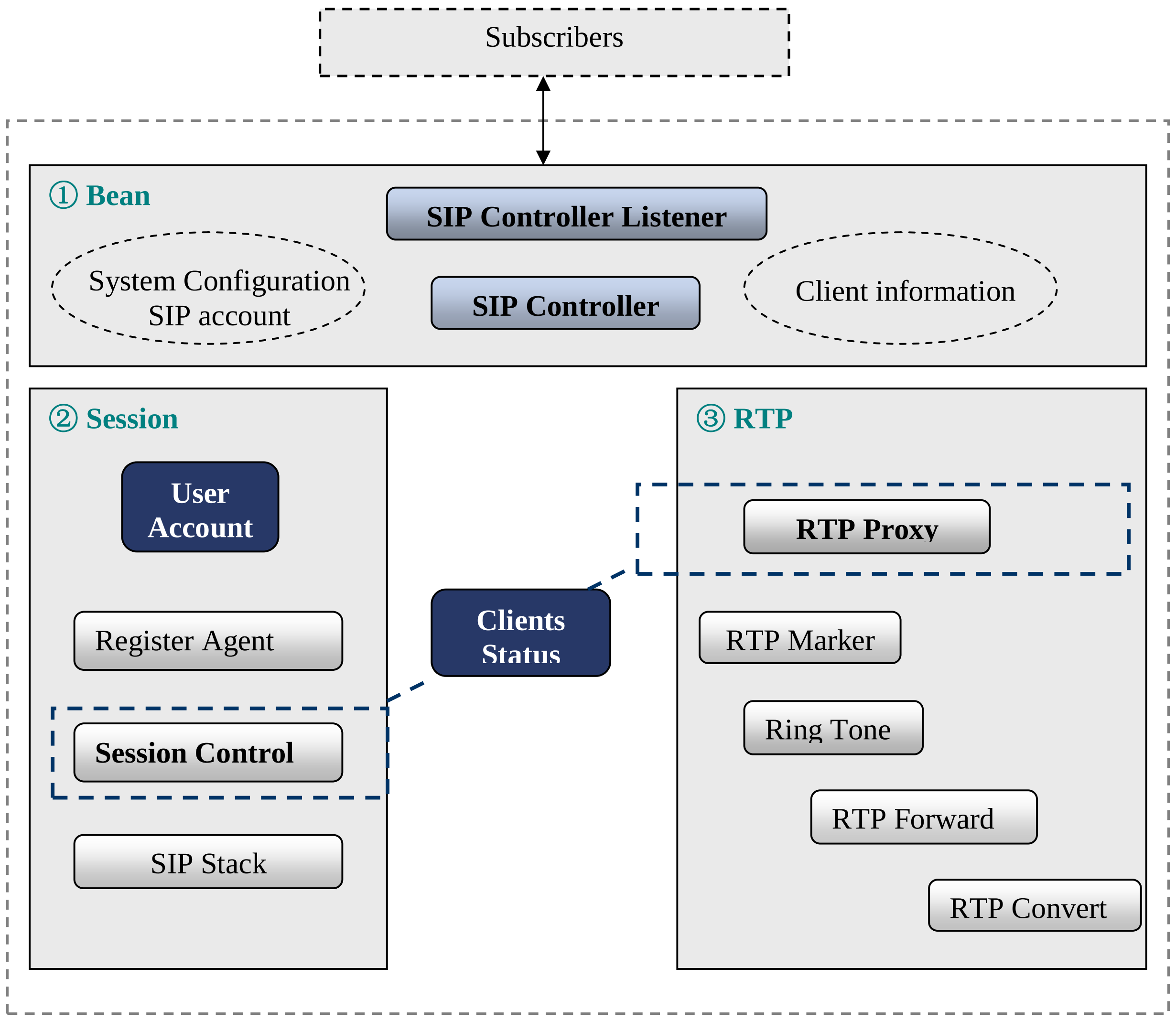


Figure 8 Design architecture of the Call Control Component

As shown in Figure 8, the Call Control Component consists of three modules:

1. SIP controller bean
2. SIP registration & session control
3. RTP proxy

SIP controller bean

The *SIP Controller* is a Java Bean class. It consists of a constructor, several getters and setters and business logic. The constructor takes key system parameters, e.g. a ring tone media file, log file path, and a port number. Users can also use the setter functions to customize more system parameters. After configuring the SIP account and client parameters, the *SIP Controller* calls the business logic methods to set up sessions.

SIP registration & session control

When the SIP registration module receives account parameters from the *SIP Controller*, it creates a *Register Agent* to handle the registration functions. The *Session Control* module is in charge of creating, modifying, and terminating sessions with participants. Moreover, the Session Control also handles the coordination between different sessions. The message sequence charts are in section 4.5.

RTP proxy

Media communication is done using the *RTP proxy*. It provides an interface for proxy-ing media packets. Four concrete sub-proxies have been implemented: *RTP marker*, *RTP Ring Tone*, *RTP forward*, and *RTP convert*. *RTP Marker* is usually sent at the beginning of the RTP transfer, acting as the starting signal of the upcoming RTP packet stream. The *RTP ring tone* proxy is actually a ring tone sender, with a user customizable ring tone media file. *RTP forward* proxy and *RTP Convert* proxy are handling the exchange of RTP packets. The *RTP Convert* proxy is called when the participants have different media encoding formats, e.g. to convert from PCMU to PCMA.

The *Session Control* and *RTP proxy* are related by the client status. *Session Control* updates the client status and notifies the proxy. This is a typical implementation of the observer design pattern, which will be described in section 4.6.4.

4.5 Session control call flows

As mentioned earlier, the session control module handles the SIP session initiation and the coordination between sessions. Due to time limitations, the Call Control Component only implements the phone-to-phone SIP call functionality. Thus, the primary operation of the Call Control Component is to establish a session between endpoints A and B while acting as a third party.

This section documents three message sequence charts that presents the three different use cases of the Call Control Component in a phone-to-phone environment.

Message Sequence Chart 1 - Normal case

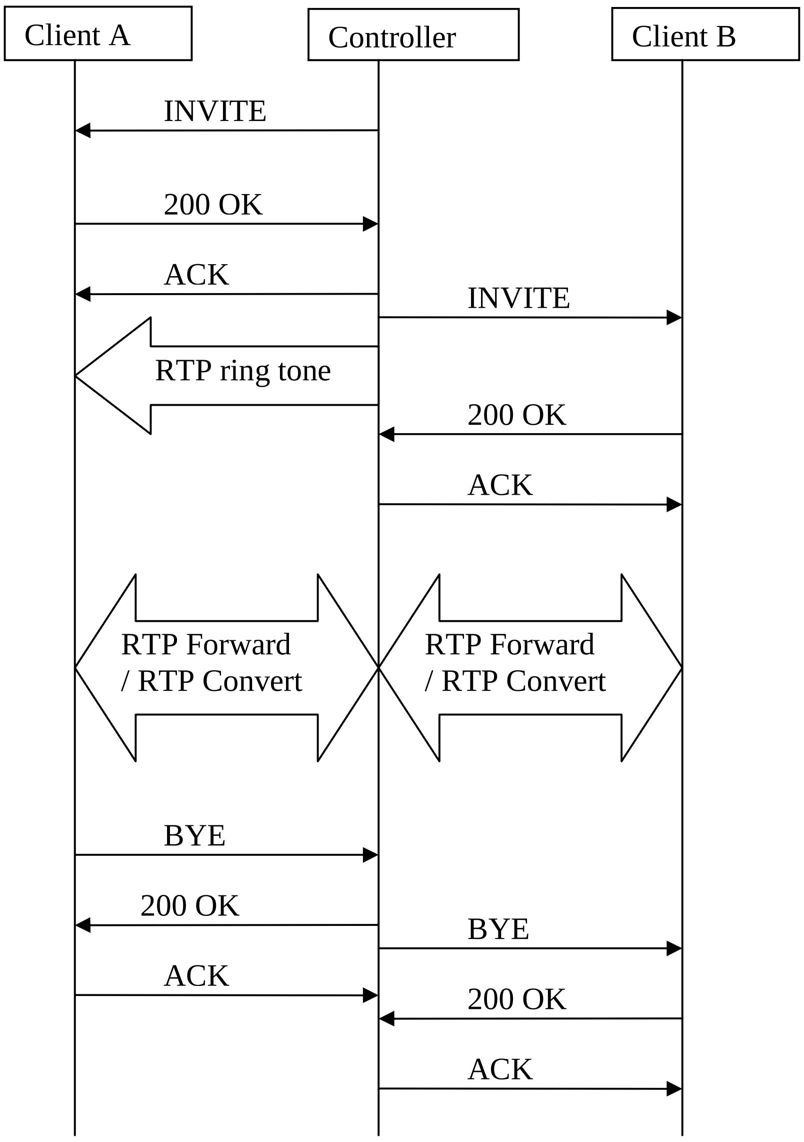


Figure 9 Message Sequence Chart – Normal case

The basic message sequence chart is shown in Figure 9. The controller first sends an INVITE to A. A's phone rings, and A answers. This results in a 200 OK that is sent to session initiator, the controller. The controller then needs to send the ACK. This part is a typical SIP session setup.

After the session between the controller and client A has been established, the controller starts to send a ring tone to client A, indicating that it is waiting for B to answer the phone.

Once the controller receives a 200 OK from client A, it start another session initiation with client B by sending INVITE to B.

After B answers the phone, the controller receives 200 OK from client B. It will stop sending the ring tone to A, and start the RTP packet proxy between A and B.

Message Sequence Chart 2 - Cancel case

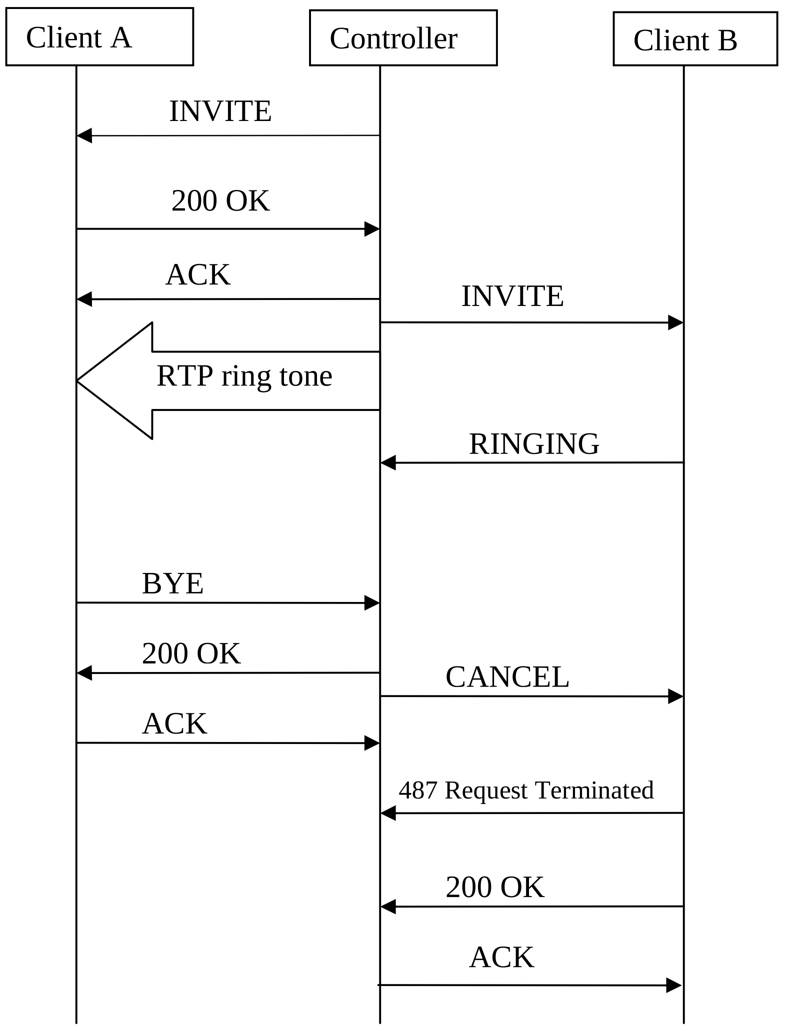


Figure 10   Message Sequence Chart – Cancel case

In the Cancel case, the session initiation with client A is the same as in the normal case. While the controller is waiting for client B to answer the phone, client A hangs up the phone. This results in a BYE to the controller. On receiving the BYE message, the controller stops sending the ring tone to A, and sends a CANCEL to B.

The CANCEL request is used to cancel the INVITE request sent by the controller. When a SIP server receives a CANCEL request for an INVITE, but has not yet sent 200 OK, it will stop sending RING, and respond to the INVITE with a specific error response “487 Request Terminated”.

Message Sequence Chart 3 – Rejection case

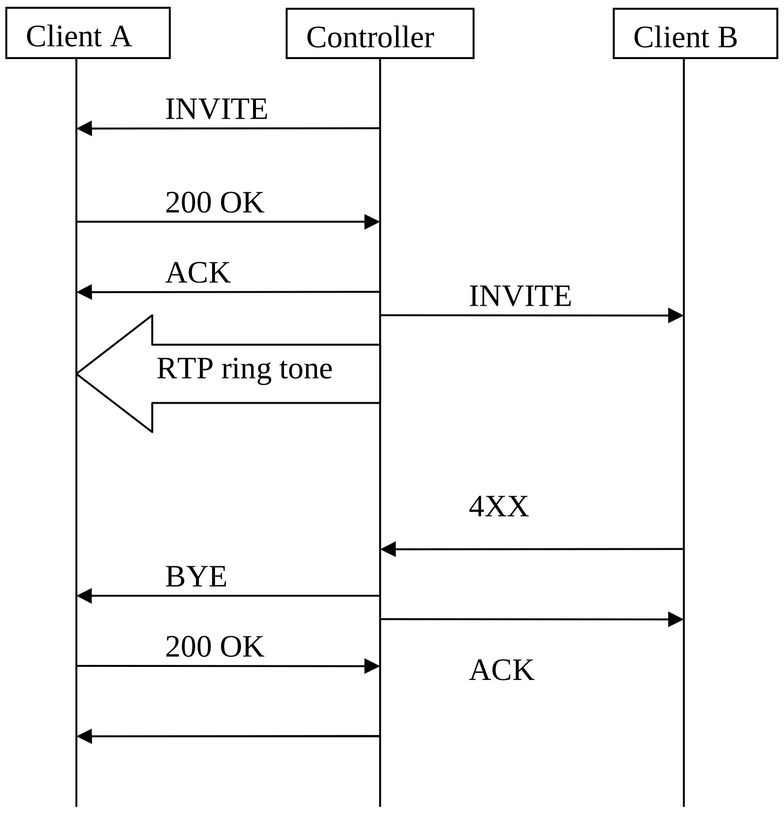


Figure 11   Message Sequence Chart – Rejection case

In the rejection case, the session initiation with client A is the same as in the normal case. While the controller is waiting for client B to answer the phone, it receives a 4xx message from B, for example, “486 Busy Here” or “404 Not Found”. In this situation, the controller will send an ACK to client B, stop sending the ring tone, and try to terminate the session with client A by sending a BYE message.

4.6 Internal design outlines

The Call Control Component design features several important design patterns. This section describes the key design choices of the component.

4.6.1 SIP Controller - Java bean

At the top of the design architecture is a Java Bean, called *SipControllerBean*, which provides a high level API to users. Figure 12 contains the sequence diagram of the SipControllerBean and related modules.

On receiving the system parameters and SIP account parameters from the user, it creates an *Account* instance to handle the account related properties and operations. When the register function is called, the account will create a *RegisterAgent* instance, which manages the SIP registration.

After a successful registration, the *SipController* calls the session logic handler-*SessionPhone2PhoneLogic* to set up session between clients. The client parameters are set with the setter functions in the *SIPController* bean.

At the end of the session, the session logic handler informs *SipController* about the end call status. The controller will then perform the un-registration operation via *Account* and *Register Agent*.

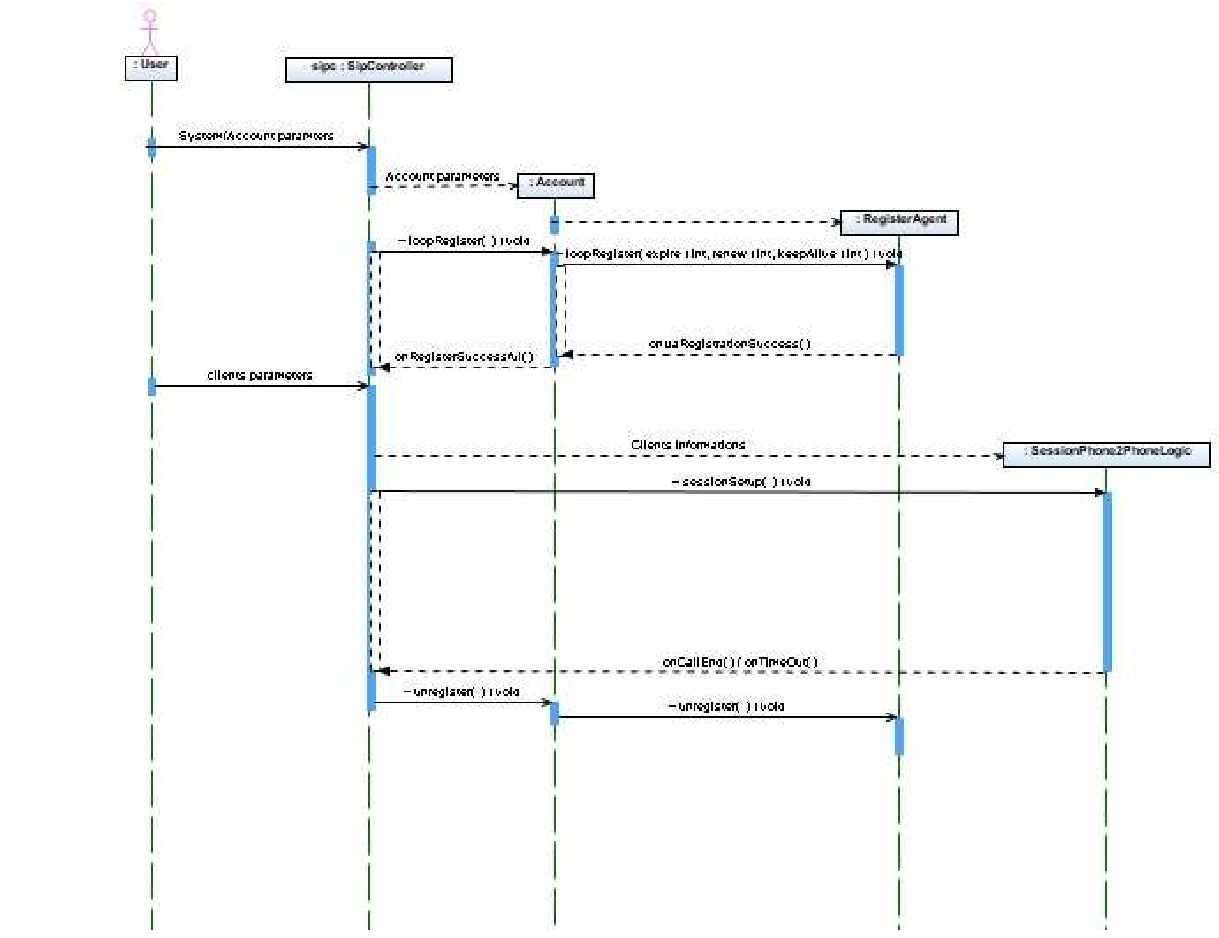


Figure 12 Sequence diagram of SIPController

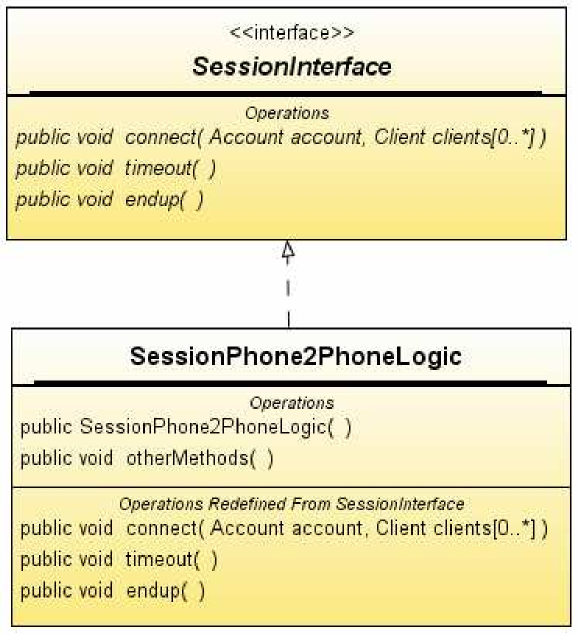
The structure of SipControllerBean is as follows:

- Constructor for essential system parameters
- Getters and setters for SIP account parameters
- Setters for other system parameters

4.6.2 Session logic control – Façade

The session logic control is below the *SipController* in the architecture of the Call Control Component. It manages the establishment and coordination of the sessions. In this library, *SessionInterface* provides an interface for developers to implement session control sub-systems. *SessionPhone2PhoneLogic* is an implementation of the phone-to-phone logic control sub-system. Developers can also implement other functionalities to extend the basic features of the Call Control Component, with, for example, conference management.

SessionPhone2PhoneLogic represent a sub-system using the Façade design pattern. A façade pattern provides a simplified interface to a larger body of code, such as sub system. Figure 13 shows the class diagram of the session logic control module.



<<interface>>
SessionInterface
Operations
public void connect(Account account, Client clients[0..*])
public void timeout( )
public void endup( )
SessionPhone2PhoneLogic
Operations
public SessionPhone2PhoneLogic( )
public void otherMethods( )
Operations Redefined From SessionInterface
public void connect(Account account, Client clients[0..*])
public void timeout( )
public void endup( )

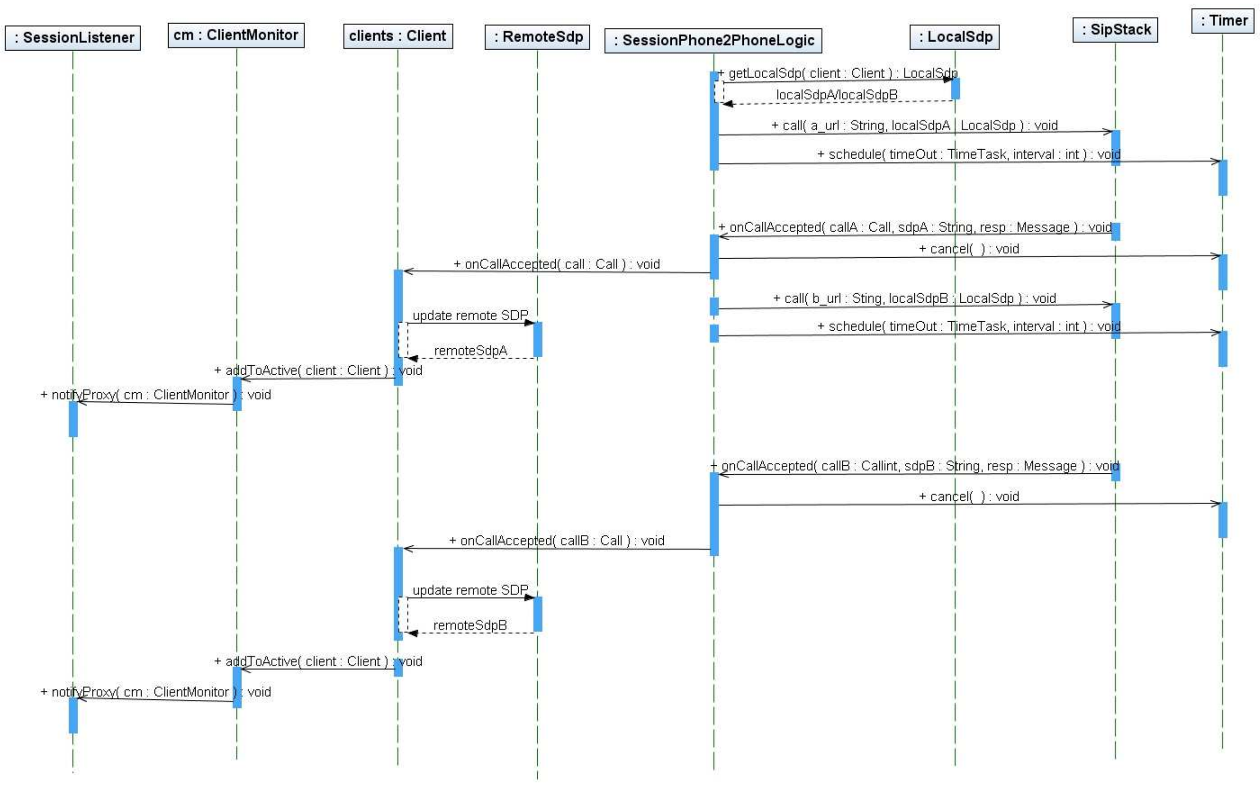
Figure 13 Class diagram - Façade pattern

The advantage of the façade pattern is:

- makes a software library easier to use and understand, since the façade has convenient methods for common tasks;

- makes code that uses the library more readable, for the same reason;

- reduces dependencies of outside code on the inner workings of a library, since most code uses the façade, thus allowing more flexibility in developing the system;



: SessionListener
cm : ClientMonitor
clients : Client
: RemoteSdp
: SessionPhone2PhoneLogic
: LocalSdp
: SipStack
: Timer
+ getLocalSdp(client : Client) : LocalSdp
localSdpA/localSdpB
+ call(a_url : String, localSdpA : LocalSdp) : void
+ schedule(timeOut : TimeTask, interval : int) : void
+ onCallAccepted(callA : Call, sdpA : String, resp : Message) : void
+ cancel( ) : void
+ onCallAccepted(call : Call) : void
+ call(b_url : Sting, localSdpB : LocalSdp) : void
update remote SDP
+ schedule(timeOut : TimeTask, interval : int) : void
remoteSdpA
+ addToActive(client : Client) : void
+ notifyProxy(cm : ClientMonitor) : void
+ onCallAccepted(callB : Callint, sdpB : String, resp : Message) : void
+ cancel( ) : void
+ onCallAccepted(callB : Call) : void
update remote SDP
remoteSdpB
+ addToActive(client : Client) : void
+ notifyProxy(cm : ClientMonitor) : void

Figure 14   Sequence diagram of session control

Figure 14 shows the sequence diagram inside the session logic control module. This is the normal case, corresponding to session control call “Message Sequence Chart 1 – Normal case”.

SessionPhone2phoneLogic initializes local SDP for each client, and calls *SipStack* to make the phone calls. *SipStack* here represents the open source SIP stack library.

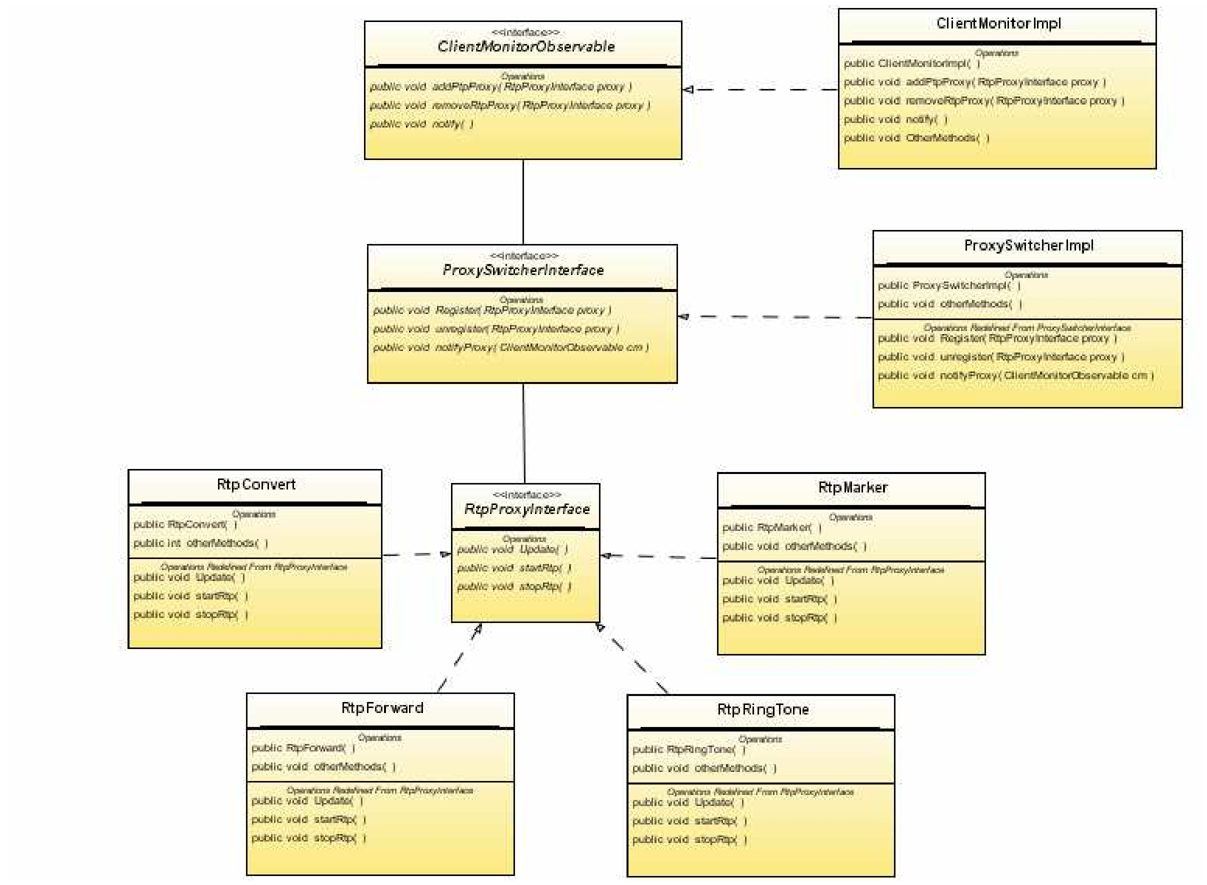
Client updates the *RemoteSDP* for each client with the SDP information contained in their responses from the endpoints. A *Client* instance manages the status and data of the client.

ClientMonitor is a monitor, which listening to all the *Client* instances. It updates the status of the *Client* pool, and notifies the *SessionListener* of any updates. *ClientMonitor* is a subject in the Observer design pattern, and is related to the RTP proxies.

Timer is to handle the timeout situations. The timeout interval value can be configured in *SipControllerBean*. The default value is 60 seconds. When an INVITE request is sent out, the Timer schedules a Timeout task, which will be canceled if a proper response is received, otherwise, it will call the timeout handler.

4.6.3 RTP proxy – Observer

The implementation of the RTP proxy is a typical Observer design pattern with a Change-Manager. Figure 15 shows the structure of the RTP proxy module.



<<interface>>
ClientMonitorObservable
public void addRtpProxy(RtpProxyInterface proxy)
public void removeRtpProxy(RtpProxyInterface proxy)
public void notify( )
ClientMonitorImpl
public ClientMonitorImpl( )
public void addRtpProxy(RtpProxyInterface proxy)
public void removeRtpProxy(RtpProxyInterface proxy)
public void notify( )
public void OtherMethods( )
<<interface>>
ProxySwitcherInterface
public void Register(RtpProxyInterface proxy)
public void unregister(RtpProxyInterface proxy)
public void notifyProxy(ClientMonitorObservable cm)
ProxySwitcherImpl
public ProxySwitcherImpl( )
public void otherMethods( )
public void Register(RtpProxyInterface proxy)
public void unregister(RtpProxyInterface proxy)
public void notifyProxy(ClientMonitorObservable cm)
<<interface>>
RtpProxyInterface
public void Update( )
public void startRtp( )
public void stopRtp( )
RtpConvert
public RtpConvert( )
public int otherMethods( )
RtpMarker
public RtpMarker( )
public void otherMethods( )
RtpForward
public RtpForward( )
public void otherMethods( )
RtpRingTone
public RtpRingTone( )
public void otherMethods( )

Figure 15   Class diagram – Observer pattern

ClientMonitorObservable acts as the subject role in the Observer pattern. It monitors the status of the clients, for example, a change in the active client list. The main functions are: *addRtpProxy(RtpProxy proxy)*, *removeRtpProxy(RtpProxy proxy)*, and *notify()*. The interface is implemented by *ClientMonitorImpl*.

RtpProxyInterface acts as the observer role in the Observer pattern. It is implemented by four sub–proxies: *RtpMarker*, *RtpRingTone*, *RtpForward*, and *RtpConvert*. The RTP proxy has one observer related method – *update()*.

RtpProxySwitcherInterface acts as the Change-Manager. When the dependency relationship between subjects and observers is complex, a Change-Manager is needed to maintain these relationships. In this case, the purpose of the proxy switcher is to minimize the work required for proxies to manage several interdependent subjects.

In the proxy module, the proxy switcher has three main responsibilities:

- It maps a subject to its observers and provides an interface to maintain this mapping
- It defines a complex and particular update strategy
- It updates all dependent observers at the request of a subject

For example, in the phone2phone logic, when the client list is updated from one active client to two active clients, the client monitor notifies the switcher about this change. The switcher then decides to stop the *RtpRingtone* proxy, and start either *RtpForward* or *RtpConvert* according to the client's media format.

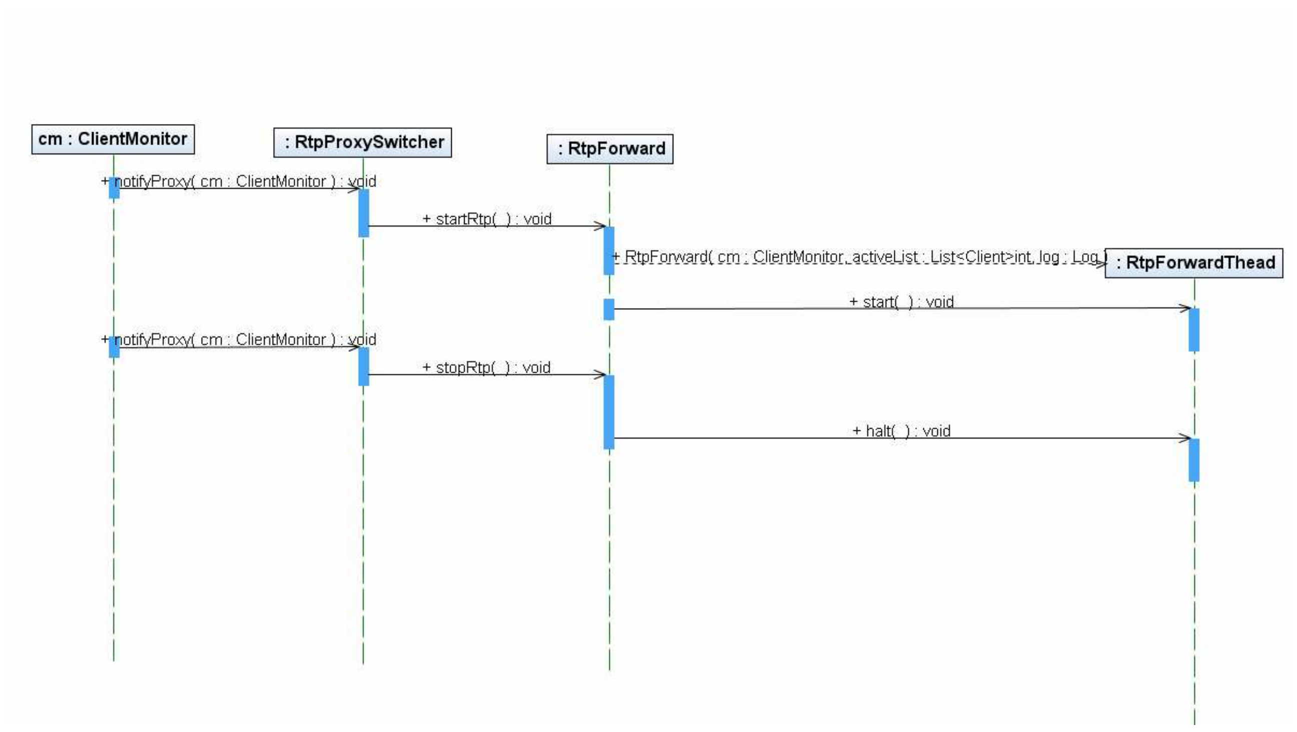

cm : ClientMonitor
: RtpProxySwitcher
: RtpForward
: RtpForwardThead
+ notifyProxy(cm : ClientMonitor) : void
+ startRtp( ) : void
+ RtpForward(cm : ClientMonitor, activeList : List<Client>int, log : Log)
+ start( ) : void
+ notifyProxy(cm : ClientMonitor) : void
+ stopRtp( ) : void
+ halt( ) : void

Figure 16 Sequence diagram of RTP proxy

Figure 16 shows the sequence diagram of the RTP proxy module. *RTP Forward* is used as an example in this figure. When *ClientMonitor* notifies *RtpProxySwitcher* about changes in the client status, it will rearrange the proxy controlling and decide whether to start or stop the sub-proxies. When an *RtpForward* proxy is started, it will create one or more *RtpForwardThread* instances and start running them. When the *stopRtp()* function is called, the *RtpForward* proxy will *halt()* the running threads.

Each thread is a one-to-many distribution model:

- One receiver: one RTP socket receives all the incoming RTP packets
- Many senders: multiple RTP sockets send out the packets that are received by the receiver

Each RTP socket corresponds to one *DatagramSocket*, which is connected to a remote socket address.

4.7 Function evaluation

This section describes the evaluation results of SIP call control component. The evaluations are basically focusing on the extendibility, transparency, efficiency and compatibility.

4.7.1 Extendibility

Extendibility is an important design goal of the Call Control Component. As mentioned in the previous sections, it offers interfaces for session control and RTP proxy management. Moreover, the RTP proxies support media communication between more than two clients. Thus, the Call Control Component can easily be extended with other functionality, for example, conferencing.

Conferencing extension

Conferencing session control could be implemented as a sub-system adapted to *SessionInterface*. Take the *connect* function for example, *public void connect(Account account, Client[] clients).*

It could be implemented by supporting a client list with more than two clients. Conferencing RTP transferring could be implemented by adding more sub proxies or simpler, modifying the *RtpProxySwitcher* to handle the intermediate logic control between client status and RTP proxies.

Example

ClientMonitor notifies proxies that the active list of clients contains A, B, C, and D. A and B have the same RTP encoding format, say PCM-ULaw, while C and D have PCM-ALaw format.

The *ConferenceSwitcher* calls the *RTP Forward* proxy and the *RTP Convert* proxy. They will create two *RTP Forward* threads and four *RTP Convert* threads.

Forward thread (1) : A (PCMU) – B (PCMU)

Forward thread (2) : C (PCMA) – D (PCMA)

Convert thread (1) : A (PCMU) – C & D (PCMA)

Convert thread (2) : B (PCMU) – C & D (PCMA)

Convert thread (3) : C (PCMU) – A & B (PCMA)

Convert thread (4) : D (PCMU) – A & B (PCMA)

However, the update function has not yet been implemented due to time limitations. Its function would be to handle real-time updates according to the changes in the client status.

4.7.2 Transparency

The transparency property is evaluated in different contexts. The most obvious transparency property of the Call Control Component is telecommunication transparency.

Telecommunication transparency

The controller is mostly transparent to the two end-points in the whole process of the session. The following figure shows the evaluation of each step in the normal-case conversation. This also applies to other conversation cases such as rejection and cancel.

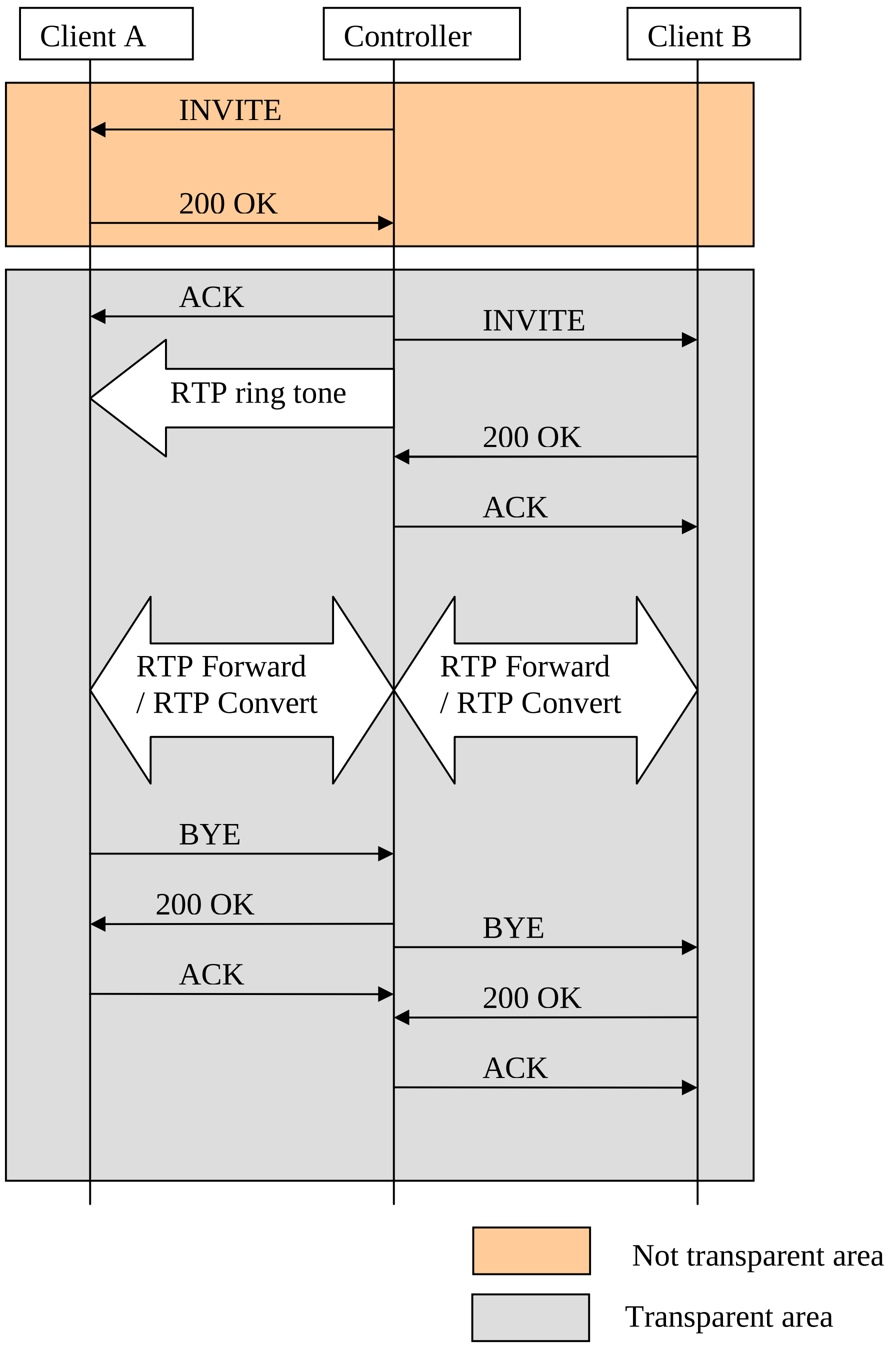


Figure 17 Transparency evaluation (normal case)

The controller is not transparent at the beginning of the conversation, when the controller initializes a session with Client A. But after Client A answers the phone until the end of the conversation, the controller is 'invisible' to both Client A and Client B, because the controller is acting as a transmission system, passing signals between A and B without changing the functionality.

4.7.3 Efficiency

As previously mentioned, the Call Control Component is a third-party-call controller between two endpoints. The session between users should be efficient and the latency in the communication should be small enough to maintain call quality.

Call quality is always an important evaluation property, but since the component is actually playing the User Agent Client (UAC) role in the SIP network, the call quality is greatly dependent on the selected SIP server, which is composed of several User Agent Server (UAS).

Unfortunately the accuracy of the measurement is greatly affected by the network environment and the hand-held devices as well as many other factors. So the efficiency of the Call Control Component is difficult to present in the form of statistics.

5 Web Application Solution Description

The web application is built on the “Mobile Front Controller” architecture. The Call Control Component is integrated into the web server as several different Java EE components: a custom tag, servlets, and a web service. This section will focus on the description of the custom tag, web service, database design and other Java EE design features of the web application.

5.1 Design goal

The design goal of the web application includes:

- Support for web users. The web application should provide call functionality as well as other additional functions to users that are using web browsers.
- Support for mobile users. The web application should provide services for mobile users. There are two ways of accessing these services: through WAP or a Java ME client which connects to the web service.
- Use a broad range of Java EE components. The most important purpose of the web application is to demonstrate how to integrate call functionality into a web server as different Java EE components. The selection of the implementation methods should fit the properties of the component and the use cases.
- Security. The web application should provide secure and efficient services for users. For example, the connection for transmitting private data such as passwords should be encrypted.

5.2 MVC architecture

5.2.1 Architecture overview

The web application uses the Mobile Front Controller architecture developed at Ericsson (see Ericsson Mobile World – MFC). Figure 18 shows the architecture of the web application. It is a typical MVC design.

The model layer contains a database server, a set of beans for account management, and the Call Control Component. The control layer consists of a custom tag, web services and servlets, which are constructed using the model layer. On the view layer, the web application offers a HTML version for web browser, two XHTML-MP versions for WAP browser with different screen sizes, and a Java ME Application for more advanced mobile phones which support the "JSR 172 - J2ME Web Services Specification".

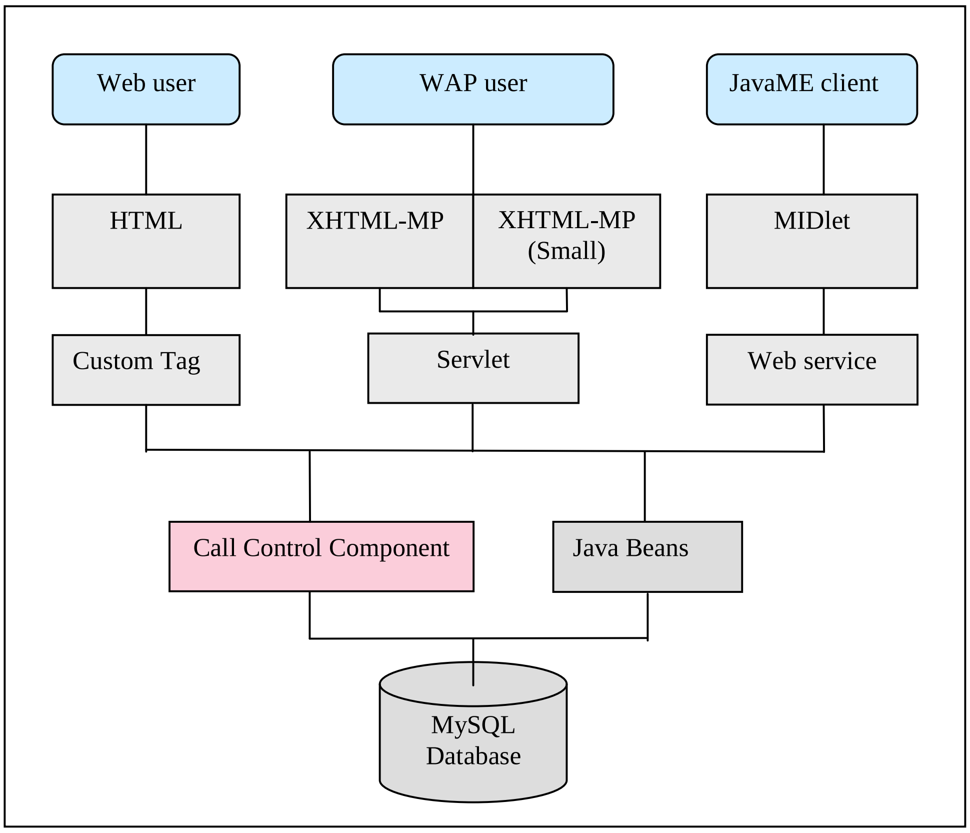


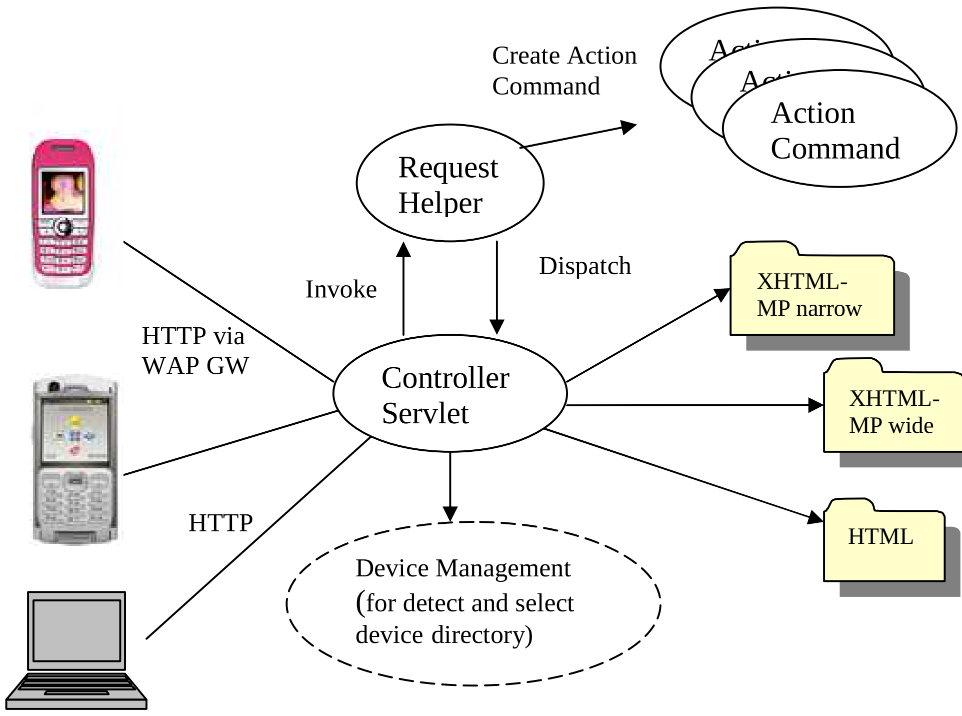
Figure 18 Architecture of the web application

5.2.2 Mobile Front Controller architecture

The web application is built using the Mobile Front Controller architecture, which is a development solution kit in the Ericsson mobile world [13]. The following is an excerpt from the documentation of the Mobile Front Controller Development Solution Kit by Peter Yeung, Ericsson AB.

Mobile Front Controller is for internet mobile applications i.e. application with combined Mobile Browsing and Web browsing with shared UI logic. This can be used for example when you develop an application for a Dot Mobi top domain. With the Dot Mobi top domain you can guarantee mobile browsing ready towards the end-users. Dot Mobi domain name is very useful when you for example marketing your mobile services towards the public end-users. This is an also way for you to make end-user find you mobile service e.g. via Google.

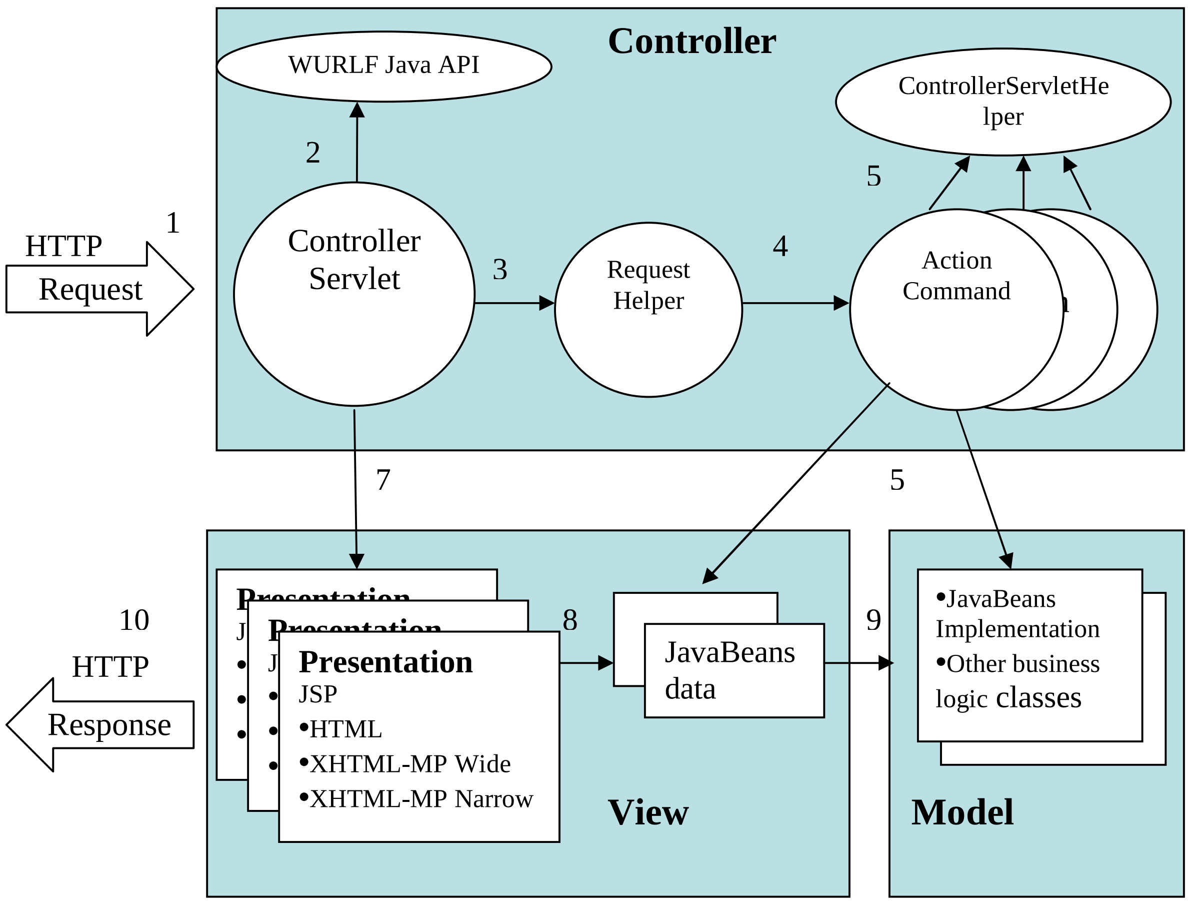
Following picture bellow shows how this solution works



Mobile Front Controller

We can dived the browsers in 3 main category i.e. Web browser desktop device uses HTML, mobile device with wide screens e.g. smart phones and the mobile browsers for narrow screens. Dot Mobi initiative has define the 2 mobile screen sizes for fit the major mobile devices on the market that we are now try to adapt in this solution.

Internal MVC architecture solution



This figure shows the MVC architecture implemented with the Mobile Front Controller solution.

5.3 Functionality

The web application contains phone-to-phone functionality by virtue of the Call Control Component. There is however also other functionality in the web application, for example, user accounts management, a click-to-call service, and administrative functions.

SIP call function

As mentioned earlier, the Call Control Component is integrated into the web application as various Java EE components such as Java Beans, servlets, a custom tag and a web service. The following table describes the different implementations and the corresponding use cases.

Use case	**Implementation method**	**View context**	**User group**
Phone-to-phone (WAP)	Servlet	WAP version	Authenticated user
Phone-to-phone (Web)	Custom tag	Web version	Authenticated user
Click-to-call	Custom tag	Web version	Anonymous
Phone-to-phone (Java ME)	Web service	Java ME Client	Java ME Client user

Account management

Figure 19 shows the functional structure of the account management as well as the database design.

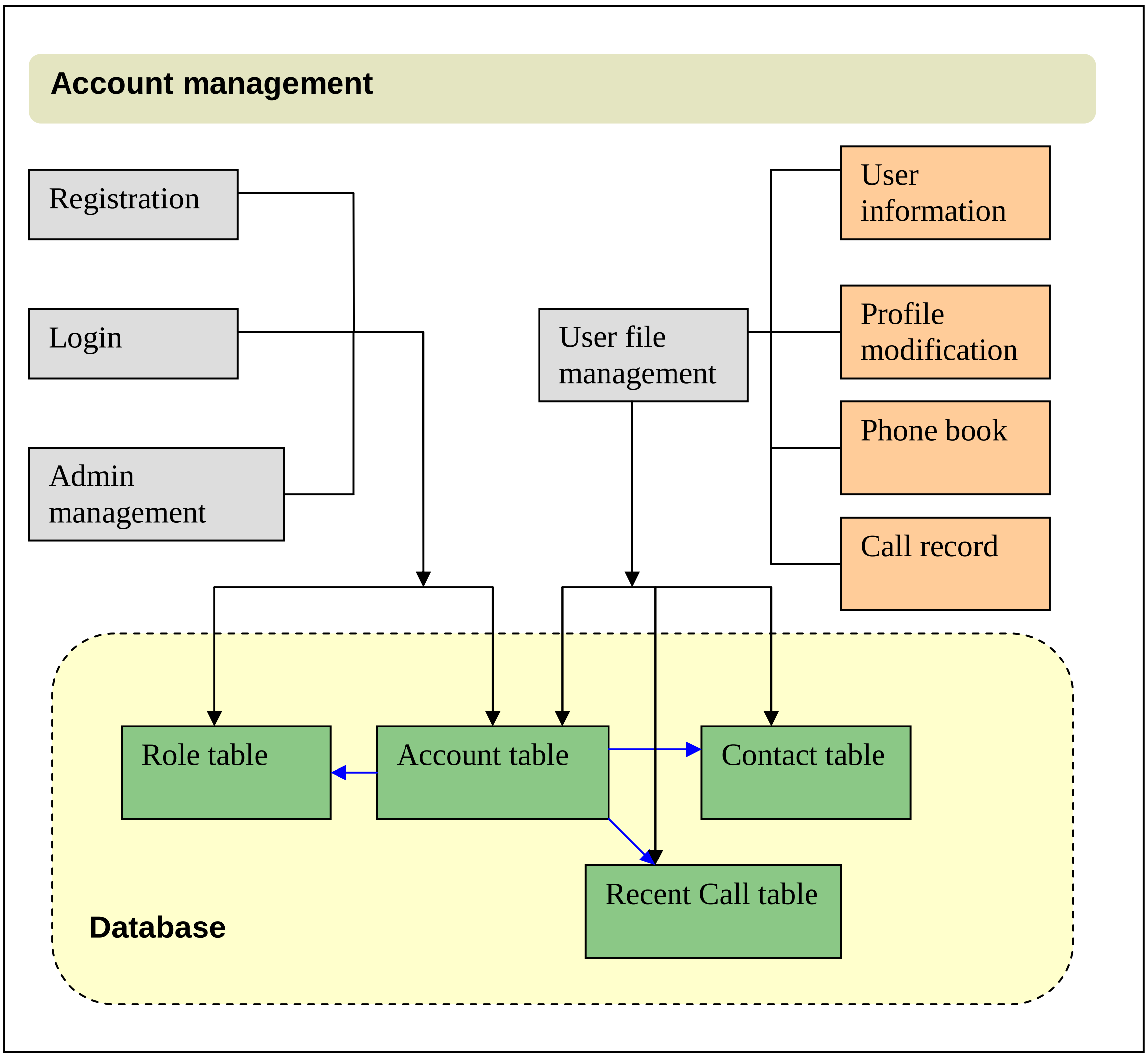


Figure 19　Functional structure of the account management

The administrator has more functions compared to other registered users. It can add and delete users.

User file management consists of four sub-tasks:

- User file information. A view of the user profile retrieved from the database.
- Profile modification. Users can edit their profiles; for example, change the SIP server, their default numbers or their password.
- Phone book. User can add, edit, and delete their contact lists.

- Call record. The web application automatically records the call list in the database when a user makes a phone call. The maximum length of the record can be adjusted.

Other functionality

Other functionality includes security protection (see section 5.3.3), and database access. The database access module provides two ways to connect the database: a JDBC direct connection and a JNDI (Java Naming and Directory Interface) connection.

The detailed use case demonstration and page flow can be found in “Appendix A – Web Call SDK developer’s guide”, Chapter 3.

5.4 Design elements

5.4.1 Phone2PhoneCall custom tag

The *Phone2PhoneCall* custom tag is part of the Click-to-dial use case. Developers can use this tag in their JSP code, input the required parameters, and call the Call Control Component to make phone-to-phone calls.

The main advantage of using custom tags is that it hides implementation details from the view layer. The execution of custom tags is the same as the JSP tags in the JSP Standard Tag Library (JSTL).

As we know, the readability and maintainability of tags are better than scriptlets. It hides the implementation details from web designers. That means the logic is better separated from the view.

Phone2PhoneCall tag handler

A tag handler is a class, and is invoked by a Web container at runtime to execute the functionality of the custom tag.

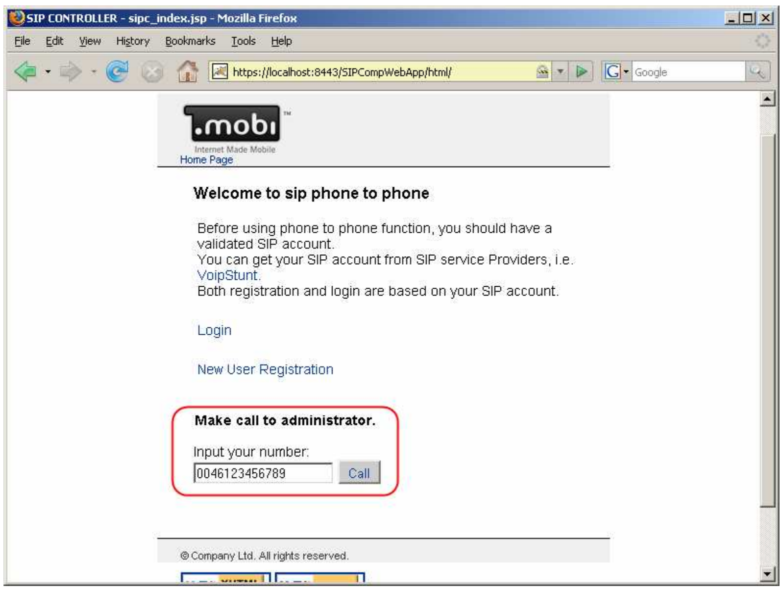
The Phone2PhoneCall tag handler extends *TagSupport*, and also implements the *SipControllerObserver* from the Call Control Component. It integrates SIP call functionality in a JSP tag by overriding the *doStartTag()* method. The main functions of the custom tag are:

- SIP registration
- Phone-to-phone calls

When the tag is loaded, it will call the Call Control Component to make phone-to-phone calls.

Click- to-call

The phone2phone tag is independent and can be used in different use cases. In the web application, there are two examples that show how to use the Phone2Phone tag: Click-to-call and phone2phone call.


SIP CONTROLLER - sipc_index.jsp - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost:8443/SIPCompWebApp/html/
Google
.mobi
Internet Made Mobile
Home Page
Login
New User Registration
Call
© Company Ltd. All rights reserved.

Welcome to sip phone to phone

Before using phone to phone function, you should have a validated SIP account.
You can get your SIP account from SIP service Providers, i.e. VoipStunt.
Both registration and login are based on your SIP account.

Make call to administrator.

Input your number: 0046123456789

Figure 20 Click-to-Call use case

The "Click-to-call" service allows users to dial a phone number, and directly talk with an administrator or a customer service representative of an organization, see Figure 20. The Click-to-call example shows the simplicity and efficiency of using the Phone2Phone tag. It also features the typical advantage of custom tags: hiding the complexity in a high level abstraction. The steps below shows the Click-to-call use case.

1. User input his own phone number, and clicks the ‘call administer’ button
2. The controller is activated by the tag. The controller calls the user’s phone
3. User answers the call, and then the controller calls the administrator’s phone.
4. Administrator answers, and the conversation begins.

The phone2phoneCall tag has some attribute parameters, i.e. *ringTone* and *userLogFile* which can be left undefined. The *SIPController* won’t produce any ring tone or user log then.

In the “Click-to-call” service, the attribute parameters a fixed number and SIP account parameters set by the administrator, to provide users with click-to-call services. These parameters are configured in a servlet context file.

5.4.2 Web service

The web service is a SOAP-based web service. It provides phone-to-phone SIP calls and other additional functions for clients.

SOAP vs REST

As we know, Simple Object Access Protocol (SOAP) is a protocol for exchanging XML-based messages over computer networks, and is usually carried over a HTTP connection.

REST (Representational State Transfer) is an architectural approach. It is resource oriented; each unique URL is a representation of some object. The retrieval and modification of resources can be done using HTTP methods such as GET, PUT, POST, and DELETE.

In this case, the SOAP-based web service is a better fit than a REST based web service. This has two main reasons:

- The web service does not contain a lot of resources. The most important resource is only one object – the Call Control Component. So a resource oriented web service would not be a good fit.
- The web service requires the server to keep state of the Call Control Component, and to report the status back to clients. Since REST based web services are supposed to be stateless, a SOAP web service is more suited choice because it allows state to be maintained.

Functionality

There are four methods in the web service:

- getRingBuffer : get the recent call records from the database for users.
- getPhoneBook: get the contact list from the database for users.
- p2pCall : make phone-to-phone calls between two clients. The client parameter is the phone number or SIP address.
- getStatus : get the call status from the SIP controller.

This SDK uses the tool wsgen from Java Metro to generate portable artifacts from an implementation class.

5.4.3 Security feature

The security of the web application is a combination of SSL certificates, security constraints, and form-based authentication.

SSL certificates

The security solution of the web application includes support for SSL certificates. This security mechanism provides end user authentication using HTTPS (HTTP over SSL). It performs mutual (client & server) certificate based authentication with a set of different cipher suites.

Security constraint

There are three user roles in the web application. The relation between roles and the resources are described in the following table.

User role	**Description**	**Accessible web-resource**	**Accessible function**
Anonymous	Anonymous user	Home page Registration collection	Click-to-call Registration
Sip-user	Authenticated user	Home page Registration collection Login user collection	Click-to-call Registration Phone-to-phone call Profile managenment Phone book management
Sip-admin	Administrator	All the pages	All the functions that sip-user has + User management

Form-based authentication

Web containers, such as Tomcat, have support for built-in security mechanisms for their applications. Tomcat offers three types of authentication mechanisms: basic, form-based, and mutual authentication.

In this web application, form-based login is used, because the login page can be customized and the authentication can be associated with security constraint or user domains.

The form-based login is using database authentication. A JDBC realm is a common solution. It involves configuring the Database Security Realm in Tomcat and initialization of user tables in the database.

5.4.4 Port control

As a server side application with network functions, the port control is an important design feature. As we know, the Call Control Component will occupy ports in the server. The functions in the servlet, custom tag and web service all require free ports. The port control should observe the status of every call, and assign and re-collect ports in real-time.

Observer of the Call Control Component

Port control is implemented in *com.ericsson.sipcWeb.sipcontrol.PortControl*. It implements *SipControllerObserver*. Figure 21 shows the sequence diagram between the *SipController* component and *PortControl*.

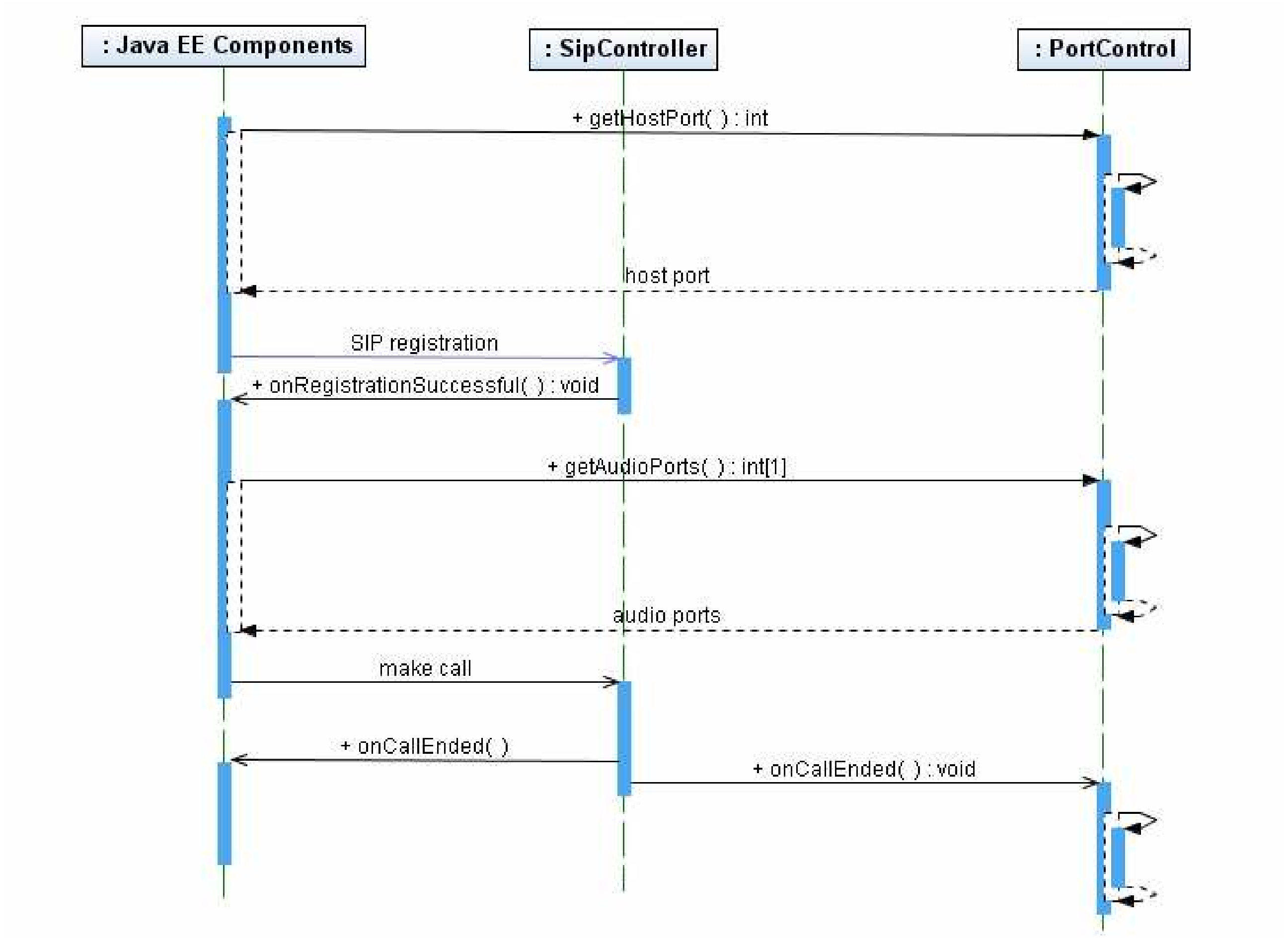


Figure 21 Sequence diagram - Port control

In this sequence diagram, Java EE components represent the *DoCall* servlet, Phone2Phone custom tag, and web service. They all use the Call Control Component to provide SIP call functionality for users.

Port control is an observer to *SipController*, and it has the following functions:

- Assign a available host port to each component at the start.
- Assign the available audio ports to the component after a successful registration.
- Re-collect the host ports and audio ports from components when the session is terminated.

- Keep track of the status of ports. Adjust the ports pool in real time.

User can manually configure the range of the unassigned ports.

Singleton in the web application

The singleton pattern is a design pattern that is used to restrict the number of instances of a class, usually one. This is useful when exactly one object is needed to coordinate actions across the system. In this case, the web application is running in a server, and the port resources should be managed by exactly one instance, so the port control should use the singleton design pattern.

As mentioned earlier, the Java EE components - DoCall servlet, Phone2Phone custom tag and web service, all need to apply for a port from PortControl, so the assignment should be integrated in one port controller.

The singleton pattern is implemented by creating a class with a static method that creates a new instance of the class if one does not exist. If an instance already exists, it simply returns a reference to that object. To make sure that the object cannot be instantiated any other way, the constructor is made either private or protected.

5.5 Future work

The web application is not only an example of showing how to integrate call functionality into a web server, but also an integral web call application with multiple functionalities. There are also some interesting features which can be implemented in the future:

Customize ring back tone. The web application can provide users with the option of using customize ring back tones. User can upload WAV files to the server, or select an existing ring back tone from the list provided by the administrator.

Multiple SIP accounts per user. The database can be extended so that each user can have more than one SIP account. The user can then select among their pre-defined accounts and choose the most appropriate one. For example, a user registers two SIP accounts. If he wants to make a call to Sweden, then he can choose the one that charges less in Sweden.

6 Java ME Client

The Java ME Client application is a client to access the web service (more details in section 5.3.2) residing in the web application described in the previous chapter. This chapter describes the design of the client and its communication with the web application.

6.1 Design goal

The purpose is to provide a convenient way for mobile users to make SIP phone calls. In addition to using WAP, mobile user can also use the Java ME client to access the web service for making SIP phone calls, retrieve their phone books and record recent calls.

As the project is targeting developers, simplicity and usability are the main design principles for the interface implementation. The following list contains the most important design goals of the client application:

- **Usability.** Besides the SIP phone-to-phone call functionality, the Java ME Client should have basic functionality similar to normal phones, for example, phone books, call records, and call status.
- **Transparency.** The users do not need to know what is going on under the interface, so all the business logic and implementation should be hidden under the view layer.
- **Simplicity.** It is usually not convenient to input data into mobile phones. Thus, the application should minimize the input requirements from the user. That means it should only require necessary information from the user, i.e. the destination number.

6.2 Java ME – web service architecture

The Java ME client uses SOAP to access the web service. The communication architecture is a typical client-server model. The client sends requests and the server responds. The main functions in the Java ME Client are: phone2phone call, phone book, and call record.

WAP users can download the JAD and JAR file of the Java ME client application after they log in to the web application. The JAD file is dynamically generated by the web application. It contains the user information and endpoint information for each user, thus greatly reducing the input required from the Java ME client. See more details in the next section. Figure 22 shows the architecture of the Java ME client and the web application.

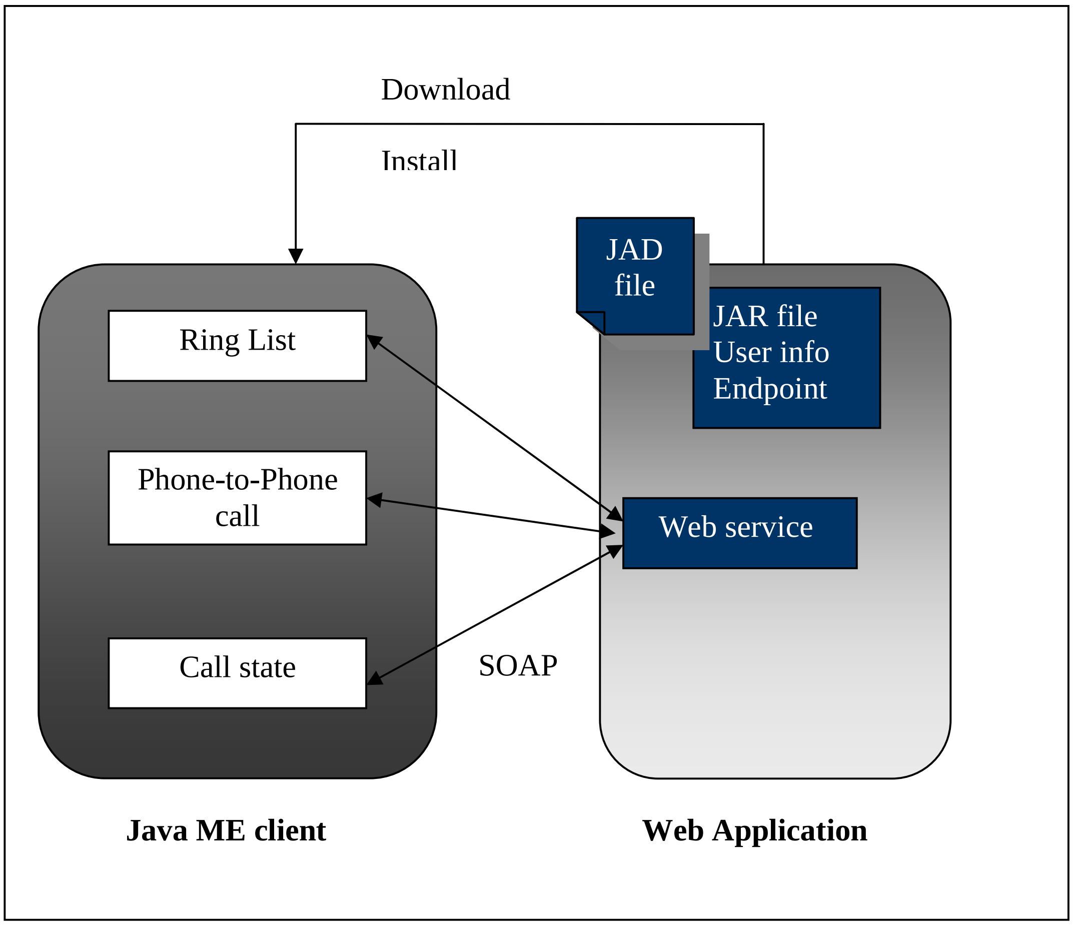


Figure 22 Architecture of the Java ME client and the web application

Communication flow

- When the Java ME client is launched, it accesses the *getPhonebook* and *getRinglist* web methods in the web service, and displays the phone book and call record in the main dialog.
- The user inputs the destination number or selects one from either the phone book or the call record, and presses the call action.
- The application accesses the *phone2phoneCall* web method in the web service, which in turn calls the Call Control Component to make phone-to-phone SIP calls.
- The destination number is recorded in the server database.
- The calling status is retrieved via the *getStatus* web method.

Application structure

The client application consists of:

- A Midlet
- Web service stubs
- Support classes

The web service stubs are generated from the WSDL (Web Service Description Language) file of the web service. Using the stub classes, the client can create a web service stub and use the web methods.

6.3 Dynamic JAD generation

The main purpose of the dynamic JAD file is to reduce the input required from the Java ME client. To access the web service, several parameters are needed:

- SIP account information: username, password, server, and realm.

- The user’s default phone number.

- Web service endpoint URL.

Most mobile input interfaces are not as convenient as computers. One solution is to use RMS (Record Management System). It is a database system for Java ME applications, saving data in the mobile phone’s memory. However, another simpler and more efficient solution is used in this project.

6.3.1 Task analysis

As mentioned in proposal, to improve the manipulation convenience and provide a simple, user-friendly interface to mobile application users, the architecture design should ensure the input amount from the client is as little as possible.

WAP steps:

1. Users register their SIP account in the web application via the WAP interface.

2. After WAP users login, the system dynamically generates a customized JAD file for each user, containing the web service endpoint, username, user number, JAR file description, etc.

3. Users can download the Java ME application and install it on their phones.

Java ME steps:

1. Mobile users can launch the phone2phoneClient application.

2. They input the destination number in the dialing form or select the number from the phone book or the call record.

3. Make the call and get the call status.

Figure 23 shows a Hierarchical Task Analysis of the process.

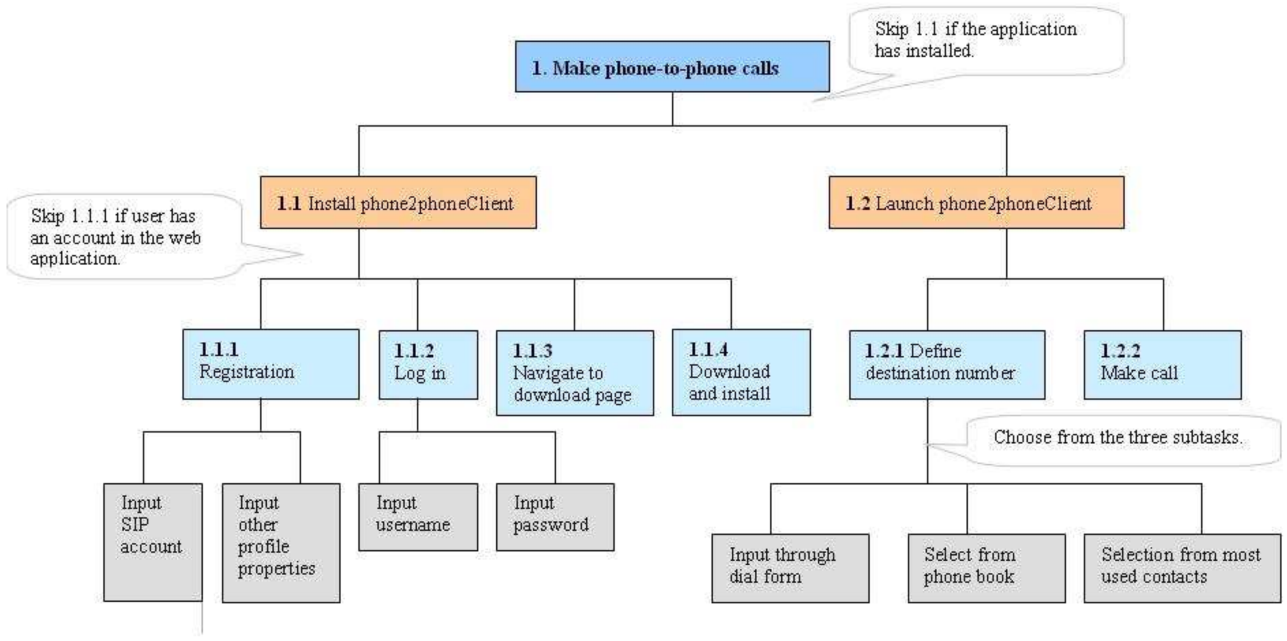

1. Make phone-to-phone calls
Skip 1.1 if the application has installed.
1.1 Install phone2phoneClient
1.2 Launch phone2phoneClient
Skip 1.1.1 if user has an account in the web application.
1.1.1 Registration
1.1.2 Log in
1.1.3 Navigate to download page
1.1.4 Download and install
1.2.1 Define destination number
1.2.2 Make call
Choose from the three subtasks.
Input SIP account
Input other profile properties
Input username
Input password
Input through dial form
Select from phone book
Selection from most used contacts

Figure 23 Hierarchical Task Analysis Tree

6.3.2 Implementation description

The web application dynamically generates the customized JAD files for WAP users after they login. The users can then download and install the phone2phone client application in their mobile phones. For details on using the client application, see Appendix A, chapter 3.

The following steps describe the process flow of this design:

1. Prepare the JAR file. Put the generated JAR file, for example, *p2pSuite.jar* into the web application so that users can download it.

2. Prepare the JAD page. To dynamically generate a customized JAD page, the web application puts the relevant parameters into *HTTPsession* attributes. For example, it calculates the size of the *p2pSuite.jar*, retrieves their *ownNumber* from the database and writes them into a JAD file. Figure 24 shows an example of a JSP page with the JAD content type.

```
<%@page contentType="text/vnd.sun.j2me.app-descriptor" pageEncoding="UTF-
8"%>MIDlet-Jar-URL: ${javaMEClient}
MIDlet-Jar-Size: ${javaMEClientSize}
MIDlet-Name: Phone2PhoneCall
MIDlet-Vendor: Ericsson
MIDlet-1: Phone2PhoneCall, , com.ericsson.p2pcall.javame.client.P2PClient
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
MIDlet-Description: Phone to Phone call service
Username: ${account.userName}
OwnNumber: ${account.defaultUri}
WebServiceEndPoint: ${WebServiceEndPoint}
```

Figure 24 Dynamic JAD page

- After a WAP user logs in, the web application dynamically initiates their customized JAD files for downloading.
- After downloading and installing the *p2pSuite.jar*, the clients can easily launch the phone2phone application and input the destination number, and make phone calls without other complicated input steps.

The security aspect of this is performed by SSL certificates.

6.4 Compatibility limitation

Although using the Java ME client is more convenient than using WAP, there are compatibility limitations to this solution.

Because the Java ME client uses “JSR 172 – Java ME Web Service Specification”, only mobile phones supporting this new Java Specification can install and use this application.

The Java ME Client has been tested successfully on several mobile phones including the SonyEricsson P990, and the SonyEricsson P1i.

6.5 User Interface evaluation

As mentioned earlier, the Java ME client is designed to function as a development tutorial for Java developers, so simplicity and usability are the main design goals for the current user interface.

But the Java ME client can also be as a source for a commercial product. The current user interface is lacking in interactivity and attractiveness. The download and installation process are also relatively complex, thus a new user interface should be developed for commercial use. The following sections outline some possible user interface improvements.

6.5.1 Improvements

WAP interface simplification

In the current design, users can register their SIP accounts and login via the WAP interface, and then download and install the customized JAD file. This solution is to reduce the input required from mobile users. Thus, users only need to input/select the destination number to make SIP phone calls, because the SIP account and Web Service endpoint are already defined in the system properties. This feature is maintained in the improved design, but there are some modifications to the WAP interface.

The current design of the WAP interface is complicated. It takes several steps for a new user to install the phone2phoneService application. Since the design should always focus on actual usage, the registration, login, download and installation should be as simple as possible. The simplifications are listed below:

- Combine the registration, login, and download page into one form. This means that the registration form requires new users to input their SIP account information (username, password, SIP server, etc.) and their own phone number as the default number, and then press the "Install" button, which redirects to the customized JAD file based on the parameters, and automatically downloads and installs the JAR file on the user's cell phone.

- No data is stored on the server. If the user wants to update/modify their information, they just need to redo the processes again and update the phone2phoneService application.

Tree-like menu design

One of the design goals of the phone2phone application is to simulate the interface of normal phones, the Apple iPod and other hand-held devices. One of the goals of the design is to keep the interface consistent with what the user already knows and is used to on that platform [14], so the tree menu does not change a lot from the one described in the HTA diagram. The following are some standardizations and improvements:

- Scrolling in all menu dialogs wraps around (if the end of a menu is reached, pressing down or up on the phone moves the selection to the first or last item depending on the original position of the cursor).

- The menu structure is shown below:

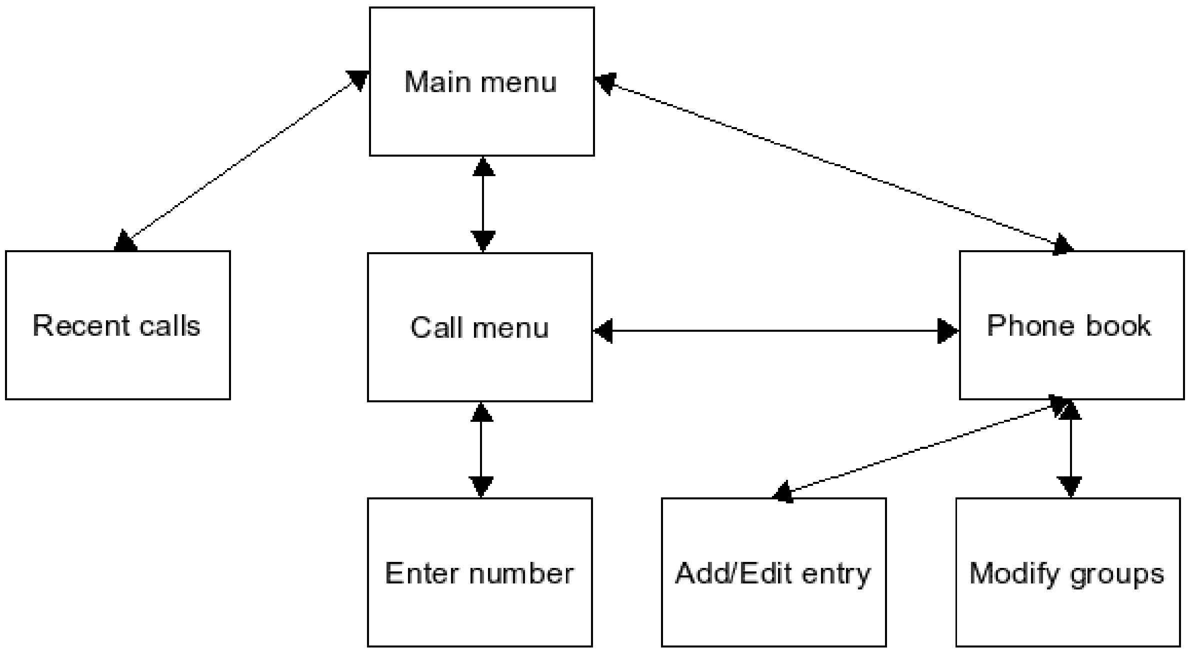


Figure 25 The menu structure of the Java ME Client

7 Conclusion

The Web Call SDK presents an innovative solution for integrating SIP call functionality into web servers. It not only meets the requirements set out in the thesis specification, but also extends it with additional functionality.

The Web Call SDK provides a Call Control Component library with a high-level API for developers. The Call Control Component is an integral component with several essential functions: SIP session control, RTP media proxy and ring back tone. This solution is an alternative to 3PCC, but with better compatibility to the current SIP networks. The Call Control Component is light-weight, extendible, efficient, reusable, and transparent.

The Call Control Component is integrated into a web application in the form of several Java EE components. The web application presents three main SIP call use cases: Click-to-Call, phone-to-phone, and a Java ME Client. Other features include: MVC architecture, SSL security, port control, etc. The web application also gives examples of building a secure and efficient web application for both Web and mobile users. The Java ME Client provides another way for mobile users to access SIP call function, other than WAP.

The Web Call SDK has been selected as Ericsson's presentation program at the JavaOne conference, 2008, with as session title “Mobile End-to-End Communication Services with Java ME and Java EE” (Session ID: TS-5802). [15]

8 Future work

There are some tasks and features in the Web Call SDK that are left for future development. Below is a list of these features:

Conferencing control: the SIP call component can be extended with a conference call feature.

Third Party Call Control (3PCC): if 3PCC can be put into practice, the load on the controller server side could be greatly reduced, because once the session is set up between endpoints, the controller only needs to manage the SIP session state. The RTP transfer would then be peer to peer between endpoints. Unfortunately, most SIP servers and gateways don't support the 3PCC specification yet, which make it impossible to implement 3PCC at the moment. However SISCO has recently released SIP gateways providing support for 3PCC, so there is hope that this feature will be supported in the near future.

Customize ring back tone. The web application can provide users with the option of using customized ring back tones.

Multiple SIP accounts per user. The database can be extended so that each user can have more than one SIP account. The user can then select among their pre-defined accounts and choose the most suitable one.

9 References

[1] Skype web site, http://www.skype.com/intl/en/, visited 2008-04-12.

[2] VoipStunt web site, http://www.voipstunt.com/, visited 2008-04-12.

[3] IETF, RFC3261 - SIP: Session Initiation Protocol, June 2002.

[4] Javvin Technologies, Voice Over IP and VOIP Protocols, http://www.javvin.com/protocolVOIP.html, visited 2008-05-05.

[5] Sun, Java EE Introduction, http://www.javvin.com/protocolVOIP.html, visited 2008-03-15.

[6] Olle Eriksson, Software Architecture with Java, http://www.it.uu.se/edu/course/homepage/pvarkjava/vt08/, visited 2008-02-12.

[7] Roy Thomas Fielding, Architecture Styles and the Design of Network-based Software Architectures, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, visited 2007-12-20.

[8] Sun, Java ME Introduction, http://java.sun.com/javame/index.jsp, visited 2008-03-05.

[9] Sun, JSR 172 (Java ME Web Service APIs), http://java.sun.com/products/wsa/, visited 2008-02-15.

[10] IETF, RFC 3725 - Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP), http://tools.ietf.org/html/rfc3725, visited 2008-09-01.

[11] Luca Veltri, University of Parma, MjSip toolkit, http://www.mjsip.org/mjua.html, visited 2007-10-11.

[12] Asterisk, http://www.asterisk.org/, visited 2008-04-26.

[13] Ericsson Developer's Program, Ericsson Mobile World, http://www.ericsson.com/mobilityworld, visited 2008-05-10.

[14] Jakob Nielsen, The Need for Web Design Standards, http://www.useit.com/alertbox/20040913.html, 2004-09-13.

[15] Sun, JavaOne 2008, http://java.sun.com/javaone/sf/index.jsp, May 2008.