Service Improvements for a VoIP Provider

ZHANG LI

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:104

Service Improvements for a VoIP Provider

Master Thesis Final Report

Zhang Li
lizhang@kth.se
2009-08-28

Examiner:
Professor Gerald Q. Maguire Jr.

Supervisor:
Jörgen Steijer
Opticall AB
Stockholm, Kista, Isafjordsgatan 22

Abstract

This thesis project is on helping a Voice over Internet Protocol (VoIP) service provider by improving server side of Opticall AB's Dial over Data solution. Nowadays, VoIP is becoming more and more popular. People use VoIP to call their family and friends every day. It is cheap, especially when users are abroad, because that they do need to pay any roaming fee. Many companies also like their employees to use VoIP, not only because the cost of calling is cheap, but using VoIP means that the company does not need a hardware Private Branch eXchange (PBX) -- while potentially offering all of the same types of services that such a PBX would have offered. As a result the company can replace their hardware PBX with a powerful PC which has Private Branch eXchange PBX software to connect all the employees and their VoIP provider.

At the VoIP provider's side, the provider can provide cheap calls for all users which are connected by Internet. The users can initialize and tear down a session using a VoIP user agent, but how can they place a VoIP call from a mobile device or other devices without a VoIP user agent? Users want to place cheap VoIP call everywhere. VoIP providers want to provide flexible solution to attract and keep users. So they both want to the users to be able to place cheap VoIP call everywhere. Although VoIP user agent are available for many devices as a software running on a computer, a hardware VoIP phone, and even in some mobile devices. However, there are some practical problems with placing a VoIP call from everywhere. The first problem is that not every device can have a VoIP user agent. But if you do not have a VoIP user agent on your device, then it would seem to be difficult to place a VoIP call. The second problem is that you have to connect to a network (probably Internet) to signal that you want to place a call. Thus at a minimum your device has to support connecting to an appropriate network. If your device is connecting to a mobile network, you can send signaling to set up a VoIP call through General Packet Radio Service (GPRS). However, the bandwidth and delay of the GPRS networks of some mobile operators is not suitable for the transfer of encoded voice data, additionally, some mobile operators charge high fees for using GPRS. All of these problems make placing VoIP calls via a mobile device difficult. However, if your mobile device has a VoIP user agent and you have suitable connectivity, then you can easily use VoIP from your mobile device

To provide a flexible solution to VoIP everywhere -- even to devices that do not or can not have a VoIP user agent, Opticall AB has designed Dial over Data (DoD) solution. By using this solution, you can place a VoIP call from your mobile device or even fixed phone -- without requiring that the device that you use have a VoIP user agent. This solution also provides a central Internet Protocol-Private Branch eXchange (IP-PBX) which can connect call to and from to Session Initiation Protocol (SIP) phones. Both individuals and companies can use this solution for call cost savings.

Max Weltz created the existing DoD solution in an earlier thesis project. This thesis [1] provides a good description of the existing DoD solution. As a result of continued testing and user feedback, Opticall AB has realized that their DoD solution needs to be improved in several area. This thesis project first identified some of the places where improvement was needed, explains why these improvements are necessary, and finally designs, implements, and evaluates these changes to confirm that they are improvements. An important result of this thesis project was a clear demonstration of improvements in configuration of the solution, better presentation of call data records, correct presentation of caller ID, and the ability to use a number of different graphical user interfaces with the improve DoD solution. These improvements should make this solution more attractive to the persons who have to maintain and operate the solution.

Sammanfattning

Detta examensarbete behandlar förbättringar i serversidan av OptiCall ABs “Dial over Data” (DoD) lösning som tillhandahålls för tjänsteleverantörer av VoIP. VoIP blir mer och mer populärt. Människor använder VoIP för att ringa till sin familj och vänner varje dag. Det är billigt, särskilt när användaren är utomlands, eftersom de inte behöver betala någon roamingavgift. Många företag vill också att deras anställda skall använda IP-telefoni, inte bara därför att kostnaden för att ringa oftast är lägre, utan för att bolaget kan ersätta sin traditionella företagsväxel (PBX) med en kraftfull dator som har PBX programvara för att även ansluta alla anställda till deras VoIP leverantör.

VoIP leverantören kan erbjuda billiga samtal till alla användare som också är anslutna via Internet. Användarna kan hantera VoIP samtal med en VoIP user agent, men hur kan de ringa ett VoIP-samtal från en mobil enhet eller andra enheter utan VoIP user agent? Användare vill kunna ringa billiga VoIP-samtal överallt. VoIP-leverantörer vill erbjuda en flexibel lösning för att locka och behålla användare. Även VoIP user agent finns utvecklade till många enheter som en programvara som körs på en dator, en hårdvara VoIP-telefon, och även i vissa mobila enheter. Men det finns vissa praktiska problem med att ringa ett VoIP-samtal från alla platser. Det första problemet är att inte varje enhet kan ha en VoIP user agent. Det andra problemet är att den måste ansluta till ett nätverk (troligen Internet) för att signalera att den vill ringa ett samtal. Om din enhet ansluter till ett mobilnät, kan du skicka signalerar att upprätta ett VoIP-samtal via General Packet Radio Service (GPRS). Dock är bandbredden och fördröjningen i GPRS-nät i vissa operatörers nät inte lämpliga för överföring av tal, dessutom tar vissa mobiloperatörer ut höga avgifter för att använda GPRS. Alla dessa problem gör det svårt att hantera VoIP-samtal via en mobil enhet. Men om din mobila enhet har en VoIP user agent och du har lämplig nätanslutning så kan du enkelt använda VoIP från din mobiltelefon

För att erbjuda en flexibel VoIP lösning överallt - även på enheter som inte kan ha en VoIP user agent har OptiCall AB utformad “Dial over Data” (DoD). Genom att använda denna lösning kan du initiera ett VoIP-samtal från din mobiltelefon eller fast telefon - utan att kräva att den enhet som du använder har en VoIP user agent. Denna lösning inkluderar också en central Internet Protocol-Private Branch Exchange (IP-PBX) som kan koppla samtal till och från Session Initiation Protocol (SIP) telefoner. Både privatpersoner och företag kan använda denna lösning för att minska samtalskostnader.

Max Weltz vidareutvecklade den befintliga DoD lösning i ett tidigare examensarbete. Denna avhandling [1] ger en god beskrivning av den befintliga DoD lösning. Som ett resultat av fortsatt testning samt synpunkter från användarna har OptiCall AB insett att deras DoD lösning måste förbättras på flera områden. Detta examensarbete har i första hand identifierat några områden där förbättringar behövdes, förklarat varför dessa förbättringar är nödvändiga, och slutligen utvecklat och utvärderat dessa förändringar. Ett viktigt resultat av detta examensarbete visades av en tydlig demonstration av förbättrad utformning av lösningen. Gränssnittet fick bla en bättre presentation av samtalshistorik, mer korrekt nummerpresentation. Dessa förbättringar bör göra denna lösning mer attraktivt för de personer som skall använda och underhålla lösningen.

Table of Contents

Chapter 1 Introduction 1
Chapter 2 Background 3
2.1 SIP protocol 3
2.1.1 What SIP can do 3
2.1.2 SIP Components 3
2.1.3 SIP messages 4
2.1.4 How SIP works in the DoD solution 6
2.2 Asterisk 6
2.2.1 How to configure an Asterisk server 7
2.2.2 Call Detail Records (CDRs) 10
2.2.3 Asterisk Gateway Interface 10
2.3 Java EE 11
2.3.1 EJB 11
2.3.2 JBoss 15
Chapter 3 DoD solution 17
3.1 Two methods to place VoIP calls 17
3.1.1 Call through 17
3.1.2 Call back 19
3.1.3 Other call methods 20
3.2 Caller ID 22
3.2.1 Caller ID when using Call through 22
3.2.2 Caller ID when using Call back 23
3.2.3 Caller ID when using SIP call 24
3.2.4 Caller ID when receiving Incoming call 25
3.3 Three components of the DoD solution 26
3.3.1 Symbian client 26
3.3.2 IP-PBX (Asterisk) 26
3.3.3 DoD server 28
Chapter 4 Improvements 33
4.1 Saving Asterisk configurations in database 33
4.2 Caller ID 35
4.3 Improving CDR records 36
4.4 Optimizing code 38
4.4.1 Using EJB 3 replacing EJB 2 38
4.4.2 Using model-view-controller mode 2 40
4.3 Using one GUI for both calling and configuring 41
Chapter 5 Evaluation 42
5.1 Test Environment 42
5.2 Tests for JBoss Application Server (DoD server) 43
5.2.1 Test plan 43
5.2.2 Test tool 44
5.2.3 Test results 44
5.2.4 Analysis of the test’s result’s 45
5.2.5 Problems faced in Tests 46
5.3 Test for MySQL 47
5.3.1 Test plan 47
5.3.2 Test tool 47
5.3.3 Test result 47
5.3.4 Test Analyze 48

5.4 Evaluation of Asterisk and the SIP trunk 49
5.5 Conclusions 49
Chapter 6 Conclusions and Future Work 50
6.1 Conclusions 50
6.2 Future work 50
6.2.1 Using a framework in the presentation layer 51
6.2.2 Multiple access numbers for the IP-PBX 51
6.2.3 Other ways to place call back calls (SMS and Voice) 51
6.2.3.1 SMS call back 52
6.2.3.2 Voice call back 52
6.2.4 Increasing security for call through 52
6.2.5 Providing machine readable call history 52
References 54

List of Figures

Figure 1. Simple message flow in setting up and tearing down a SIP session between A and B 5
Figure 2. Multiple call detail records when placing calls 6
Figure 3. Web application structure in JBoss 16
Figure 4. Call through procedure 18
Figure 5. Call back procedure 20
Figure 6. Showing the user's IP-PBX assigned caller ID when using call through 22
Figure 7. Showing different caller IDs when using call through between two users' connected to the same DoD system 23
Figure 8. Showing caller ID when using call back 23
Figure 9. Showing caller ID when using call back 24
Figure 10. Showing caller ID when using SIP user agent 24
Figure 11. Showing caller ID when using SIP user agent 25
Figure 12. Showing caller ID when receiving a incoming call 25
Figure 13. The web GUI for call back 29
Figure 14. The web GUI for contact book 29
Figure 15. The web GUI for administrators to manage users 30
Figure 16. The web GUI for administrator to manage the connection between Asterisk and DoD server 30
Figure 17. The web GUI for administrator to modify the numbers of IP-PBX 31
Figure 18. The web GUI for administrator to show the call history 32
Figure 19. Dialplan stored as a table in a database 35
Figure 20. Web GUI showing CDR for normal users 37
Figure 21. Web GUI showing CDR for administrator 37
Figure 22. Annotation for EJB3 39
Figure 23. Dependency Injection for EJB3 39
Figure 24. Network topology of test environment 43
Figure 25. Running users chat 45
Figure 26. Average response time for each page (in second) 46
Figure 27. Response time to query the database 48

List of Examples

Example 1. Example of a sip.conf file ... 7
Example 2. Example of an extension ... 8
Example 3. Example of context ... 9
Example 4. Example of AGI script ... 10
Example 5. Example of declaration using XML using annotation ... 14
Example 6. Example of declaration ... 14
Example 7. Example of using JNDI ... 15
Example 8. Example of using DI ... 15
Example 9. SIP account setting in Asterisk ... 34
Example 10. Dialplan saving in flat file ... 35

List of Tables

Table 1. Detail explanations of SIP messages 5

List of Abbreviations

AGI	Asterisk Gateway Interface
API	Application Programming Interface
ARA	Asterisk Realtime Architecture
AWT	Abstract Window Toolkit
CDR	Call Detail Record
CTM	Companhia de Telecomunicações de Macau S.A.R.L.
DI	Dependency Injection
DoD	Dial over Data
DTMF	Dual-Tone Multi-Frequency
EJB	Enterprise JavaBeans
GPL	GNU General Public License
GPRS	General Packet Radio Service
GUI	Graphical User Interface
Java EE	Java Platform, Enterprise Edition
JBoss	JBoss Application Server
JDBC	Java Database Connectivity
JDK	Java Development Kit
JNDI	Java Naming and Directory Interface
JPA	Java Persistence APIs
JPQL	Java Persistence Query Language
JSPs	Java Server Pages
JVM	Java virtual machine
J2SE	Java Platform, Standard Edition
IP	Internet Protocol
IP-PBX	Internet Protocol-Private Branch eXchange
ISP	Internet Service Provider
NAT	Network Address Translation
ODBC	Open Database Connectivity
PBX	Private Branch eXchange
PLMN	Public Land Mobile Network
PSTN	Public Switched Telephone Network
RTP	Real-time Transport Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMS	Short Messaging Service
SOA	Service-Oriented Architecture
TLS	Transport Layer Security
UAC	User Agent Client
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network

Chapter 1 Introduction

VoIP is a general term for a family of transmission technologies for delivery of voice communications over an Internet Protocol (IP) networks (such as the Internet or other packet-switched networks). Other terms frequently encountered and often considered synonymous with VoIP are IP telephony, Internet telephony, voice over broadband, broadband telephony, and broadband phone.[2] Note that there are subtile distinctions between these terms, but for the purpose of this thesis we will consistently use the term VoIP.

In order to transfer voice over an IP network, several protocols are widely used: SIP, H.323, and Skype.[3] SIP is the most popular and widely used protocol. Both SIP and H.323 signaling are used in combination with the Real-time Transport Protocol (RTP) for transferring the actual media session. In contrast, Skype is a proprietary protocol of which relatively few details are know, most of these by reverse engineering.[4] The Real-time Transport Protocol (RTP) defines a standardized packet format for delivering real-time audio, video, and timed text over the Internet.[5]

SIP is a signaling protocol, used for setting up and tearing down multimedia communication sessions, such as voice and video calls, over the Internet. The protocol can be used for creating, modifying, and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or more media streams.[6] The details of SIP, will be explained in section 2.1.

Since the encoded voice stream is transferred through an IP network, individual calls are very inexpensive, sometimes even free, because Internet service is generally a flat rate service and the number of bits or packets per second required to carry voice is quite limited in comparison to other types of traffic (particularly large files, images, and video). VoIP is increasingly popular, because the cost of a VoIP call between two points is lower than placing circuit-switched calls via the Public Land Mobile Network (PLMN) or Public Switched Telephone Network (PSTN). Additionally, because many VoIP providers provide connectivity between their VoIP network and the PSTN and PLMN, a VoIP user is able to place calls to both fixed and mobile phones. Depending upon the location of the voice gateway and the agreement of this VoIP provider with these other networks this cost can vary.

The easiest way to place a VoIP call is using a computer. This is because that there are many VoIP program (such as a SIP user agent or other VoIP user agent) available to use. Example of such application are: MiniSIP[7], x-lite[8], and SJPhone[9]. There are also a number of SIP service providers such as Nonoh[10], VBUZZER[11], and VOIP STUNT[12]. These service providers are interconnected with both the PSTN and PLMN. General users can call other SIP users for free, or can call users attached to the PSTN or PLMN at a low cost or even free. However, these VoIP user agents can only be used on computers or a few smart mobile phones. Only Nokia has its own SIP user agent inside their Symbian platform. There are some free SIP user agents running on mobile device such as Fring[13]. However, you cannot place a call from a mobile phone or fixed phone without a SIP user agent.

Opticall AB sells a solution called Dial over Data (DoD), which can solve this problem. The DoD solution provides a low cost telephony service for mobile phone users, fixed phone users, and users using a SIP user agent. By using the DoD solution, a user can place a VoIP call in several ways, even placing a VoIP call from mobile phone or fixed phone without a SIP user agent by splicing calls together. The complete solution consists of an IP-PBX (Asterisk[14]) and mobile phones with an optional Symbian client. The IP-PBX is connected to an upstream VoIP provider who is connected to one or more PLMNs, so the IP-PBX can allow the user to place a call to or receive a call from these PLMNs (of course the VoIP providers are also connected to one or more PSTN operators, so calls can also be delivered to or received from fixed phones). The users can use a SIP user agent, a mobile phone, or even a fixed phone to place low cost calls. In this solution (as it does not need to use a SIP user agent at the user's side), there are two main ways to make a VoIP call via the IP-PBX: "Call through" and "Call back". I will explain these two methods in section 3.1. Use of a SIP user agent, is also supported. The SIP user agent can connect to the IP-PBX which acts as a SIP proxy, in order for the user to place a VoIP call to callee.

Using SIP and RTP, many manufacturers have implemented their own VoIP hardware as well as software. Asterisk is a software implementation of a telephone PBX originally created in 1999 by Mark Spencer of Digium. Like any PBX, it allows attached caller to make calls to one another, and to connect to other telephony services including PSTN.[15] Additionally it enables calls to and from VoIP service providers.

Since Asterisk is an open source software PBX, many VoIP providers (using the SIP protocol) use this software PBX. Asterisk is free to use, hence we do not need to pay any fee for using it in commercial products. Additionally, Asterisk supports a variety of features and supports a number of devices that are of interested to us. Therefore, Opticall AB chose Asterisk as its software PBX.

This DoD solution have been developed by several persons. The complete DoD solution is up and running now, but there are some issues and some parts that need to be improved. For example, the call detail record (CDR) information is in chaos, there are two different web Graphical User Interfaces (GUIs) for controlling this solution, and caller ID is not displayed properly. One of the goals of this thesis project was to determine which parts of the existing solution needed to be improved, how they should be improved, and to implement and evaluate these improvements.

Following this initial chapter, chapter 2 provides background information about the three main technologies that are relevant to this thesis (SIP, Asterisk, and Java EE). Chapter 3 describes the current DoD solution. Chapter 4 examines the areas that need improvement and presents the improvements that have been made in each of these areas. Chapter 5 evaluates these improvements. Chapter 6 presents some conclusions about this work and suggests some future work.

Chapter 2 Background

This chapter will explain some of the technologies which are used in the DoD solution. This chapter begins with an explanation of these technologies and some equivalent technologies. This chapter also explains which parts of these technologies will be used and how they are used in the DoD solution.

2.1 SIP protocol

SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls.[16] The following sections will elaborate some of the relevant details of SIP. For a general introduction to SIP the reader is encouraged to read Sinnreich and Johnston's book: "Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol".[17]

2.1.1 What SIP can do

We normally use the SIP protocol for setting up VoIP calls. When two parties want to talk with each other, the SIP user agents send SIP messages to negotiate and establish a session. However, SIP does not carry the media content. After a session has been established, the SIP user agents use RTP to communicate - based upon the information that was passed using the Session Description Protocol (SDP) in SIP message bodies. Finally SIP terminates the session when one of the parties hangs up.

2.1.2 SIP Components

In SIP, there are several types of components. They are classified into two categories: SIP user agents and SIP servers.

- SIP user agent

  A SIP user agent could be implemented by software or hardware. There are many software SIP user agents, such as X-lite, SJPhone, and miniSIP. There are also many hardware SIP user agents (often called IP phones). By using a SIP user agent, you can establish a session to communicate with another party or parties. The SIP user agent helps you to create a session using the SIP protocol, then transfers and receives media to/from the other party/parties using RTP.

- SIP servers

  SIP user agents can work without a SIP server, as they can communicate with each other directly using their IP addresses. But in some situations, these SIP user agents cannot find each other as they do not know the other party’s current IP address. SIP servers help SIP user agents to initiate a session. There are four kinds of severs used with SIP:

  - Location server

    Used by a Redirect server or a Proxy server to obtain information about a called party's possible location. The location server is used by the Registrar server to store location information. Note that the location server is **not** a SIP server, since it does **not** utilize the SIP protocol.

  - Proxy server

    A Proxy server is an intermediary that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or transferred to other servers. A proxy interprets and, if necessary, rewrites a SIP request message before forwarding it.

  - Redirect server

    A redirect server that accepts a SIP request, maps the destination address into zero or more new addresses, then returns these addresses to the client. Unlike a Proxy server, it cannot accept calls, but can generate SIP responses that instruct the User Agent Client (UAC) to contact another SIP entity.

  - Registrar server

    A Registrar server accepts REGISTER requests. A registrar is typically colocated with a Proxy or Redirect server and may offer location services. The registrar saves information about where a party can be found.[18] Each user agent that will register with this server must have some type of trust relationship with this registrar in order to be able to register, deregister, and to protect the communications between the two.

2.1.3 SIP messages

In order to explain clearly how the SIP protocol works, we use a simple example to explain how it works. This example illustrates the traffic exchanged to set up and tear down a SIP session between SIP user agents A and B. See figure 1 and table 1.

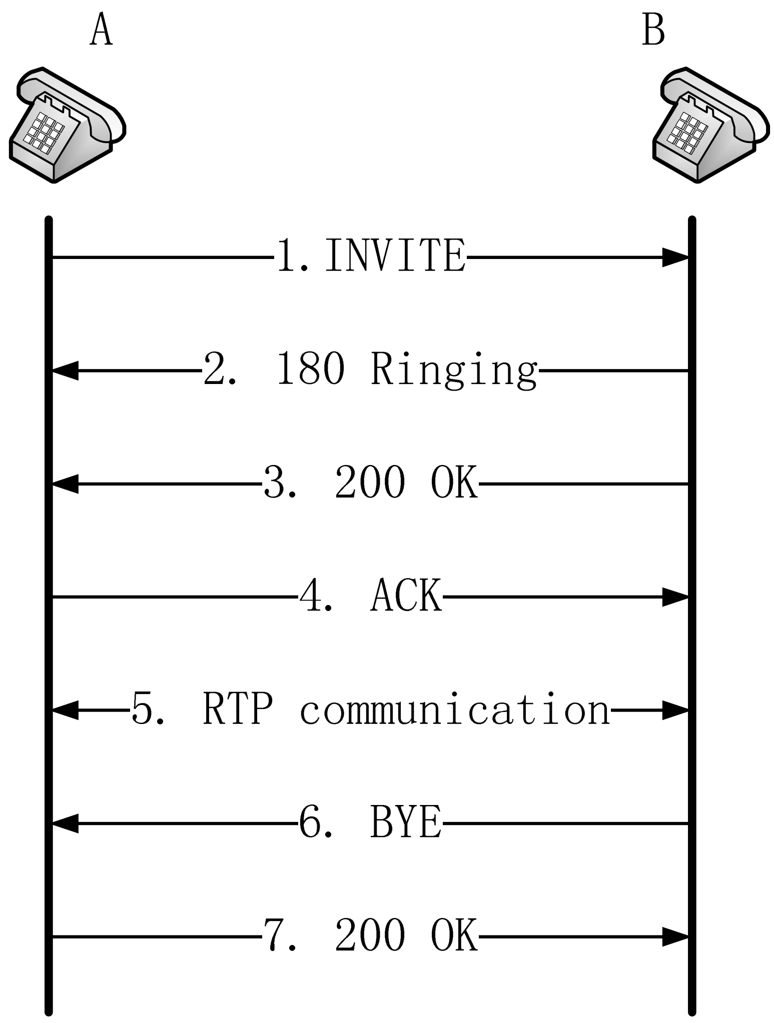


Figure 1. Simple message flow in setting up and tearing down a SIP session between A and B

SIP message	Description
1. INVITE	A sends an INVITE message to invite B to join a session.
2. 180 Ringing	After receiving the INVITE message, B starts to ring (indicating to the user that there is an incoming call) and sends back a 180 Ringing message
3. 200 OK	After B answers the phone, B sends back a 200 OK message to tell A that it is ready to communicate.
4. ACK	After A receives the 200 OK message, A sends back a ACK message and starts to transfer voice content via RTP.
5. RTP communication	Both A and B send RTP packets to each other.
6. BYE	If B wants to hang up the phone, then B sends a BYE message. After this B stops sending RTP packets.
7. 200 OK	After A receives the BYE packet, A sends back a 200 OK message, then A stops sending RTP packets.

Table 1. Detail explanations of SIP messages

2.1.4 How SIP works in the DoD solution

The DoD solution is designed to provide cheaper calls, so SIP plays a very important role in this solution. By using SIP, the cost of both placing and answering calls can be low cost. If a user calls another user, and both users are using SIP user agents, then these two users do not need to pay for a circuit switched call. If a user places a call to PSTN or PLMNs, in order to reduce the cost, the DoD solution uses a SIP trunk to place these calls. So the cost of placing such calls can also be cheaper than a single circuit-switched call. If a user answers a call which is redirected by the DoD server, then the user will be charged for this call. However, if this call is also placed using a SIP trunk, then the cost of answering this call is also cheaper (as the call can be delivered to a local gateway). Of course, the operator of the user's phone may also charge the user for making or receiving the call. It is important to note that the cost of calls using a SIP trunk depends on which SIP provider we are using. One unfortunate side effect of this method of reducing the cost of a single call, by turning it into two or possibly three separate call legs is that the call detail record now consists of two or three related records, rather than a single record, see figure 2. This leads to increased call detail record processing -- this will be addressed in section 4.3.

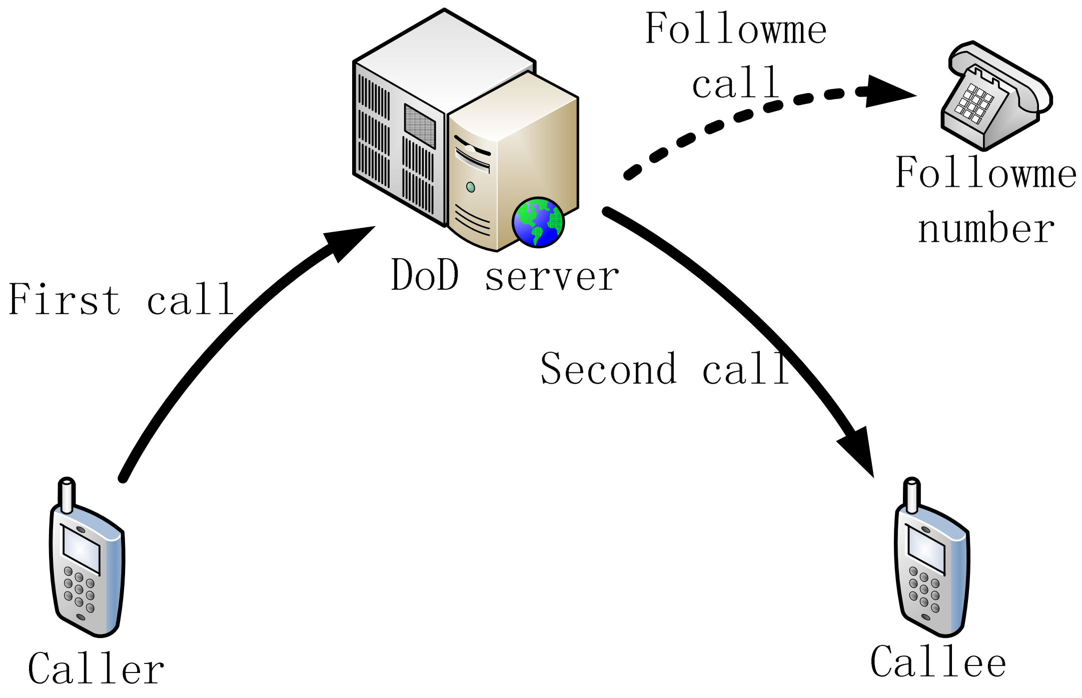


Figure 2. Multiple call detail records when placing calls

2.2 Asterisk

Asterisk is open source program. Asterisk can be used as an IP-PBX, a media gateway, and/or a call center. Asterisk supports several VoIP protocols, including the SIP protocol. Asterisk is released under a dual license model, using the GNU General Public License (GPL) as a free software license and a proprietary software license to permit licensees to distribute proprietary, unpublished system components.[15]

2.2.1 How to configure an Asterisk server

2.2.1.1 Configuration files

The configuration of Asterisk is based on configuration files. There are many configuration files which are responsible for different settings. The two main aspects of configuration are (1) to configure any phones that are to be used with this IP-PBX and (2) to define a dial plan (this defines the numbering scheme to be used within this IP-PBX).

Configuring SIP phones

One of the main reasons for using Asterisk is that Asterisk supports the SIP protocol and can utilize SIP phones as IP-PBX extensions. The configuration of these phones is specified in the file ”sip.conf”[†]. In this file, you can configure both SIP phones and SIP peers. Example 1 shows how to configure a simple SIP phone.

```
[sip-phone-1]
type=friends
host=dynamic
secret=PASSWORD
```

Example 1. Example of a sip.conf file

There are many fields that can be specified when setting up a SIP phone or SIP peer. You do not need to set every field, you only need to set fields which differ from the default setting. The first line in above example specifies a name for this SIP phone, this is simply an identifier, but this ID is very useful for an administrator to refer to this SIP phone. The following fields are very important for a SIP phone.

types Defines the type of user. There are three types:

peers	A SIP entity to which Asterisk sends calls (a SIP provider for example).
users	A SIP entity which places calls through Asterisk (A phone which can only place calls).[19]
friends	Both a peer and a user

hosts Defines the location of this SIP entity. This could be an IP address, domain name, or *dynamic.* If *dynamic* is specified, then the location of the host will be specified when the host registers with the SIP registrar.

secrets The password of the SIP entity to be used for authentication.

[†] You can find this file in the directory */etc/asterisk/*

Dialplan

The dialplan is truly the heart of any Asterisk system, as it defines how Asterisk handles inbound and outbound calls.[20] The configuration of the Dialplan is specified in the file "extensions.conf". There are several important concepts used in this configuration file, two of the most important are: Contexts and Extensions.

- **Extensions**

  In a traditional telephony system, the extension represents a specific telephone line. However, in Asterisk, an extension could be more general. An extension is a series of steps in a context. If a call matches a certain extension, then the call is sequentially processed by the logic defined for this extension. Each extension is composed of 3 parts: name, priority, and action. The name represents the extension name or number. The priority indicates the order of execution (with the highest priority operation to be performed first). The action represents the action to be taken for a call of this type for this extension. Example 2 shows an example of an extension. In this example, the extension has the number "8801" and the highest priority action is to dial, while the second highest priority is to hangup. Thus after the call is completed (successfully or unsuccessfuly) the extension will be hungup.

```
exten => 8801, 1, dial(SIP/8801)
exten => 8801, 2, Hangup()
```

Example 2. Example of an extension

- **Contexts**

  Dialplans are broken into sections called contexts. Contexts are named groups of extensions, which serve several purposes.[20] These contexts can be used to split the logics for handling calls; For example, one context might only be for outgoing calls, while another context is only for incoming calls. Example 3 shows an example of two contexts. The first of these contexts is used for calls to the number "8801" while the second is for calls to the number "8802". Note that the names of the context are purely for human consumption - the Asterisk system has no understanding of these strings (i.e., "from-sip" is a name of a context used for the convenience of the administrator who defines the dialplan and does not actually mean anything to Asterisk).

```
[from-sip]
exten => 8801, 1, dial(SIP/8801)
exten => 8801, 2, Hangup()

[incoming-calls]
exten => 8802, 1, dial(SIP/8802)
exten => 8802, 2, Hangup()
```

Example 3. Example of context

2.2.1.2 Database configuration

In Asterisk, a database is used in two ways. The first is for storing Asterisk settings. This is called the Asterisk Realtime Architecture (ARA). The second use of a database is for storing call detail records.

ARA stores the Asterisk configurations. While the configuration could be stored in files, by using ARA, you can store the configuration in one of several different kinds of databases[‡]. There are some advantages of using the Asterisk Realtime Architecture rather than configuration files. The biggest advantage is that you can design a distributed Asterisk system. The configuration of Asterisk could be stored at another host, then a number of Asterisk servers can use this shared configuration to their users' information. Note that by using a database you also have the traditional database features of roll-back (to restore the configuration to an earlier configuration) and consistency - so that the configuration (even in the distributed case) can remain consistent. Additionally applications can manipulate the configuration, for example providing a GUI for showing and modifying the set of configurations. The Asterisk Realtime Architecture could be Static or Dynamic.

- **Static ARA**
  If you use a static ARA to store configurations, you have to reload the configurations after you change the configuration. Normally a static ARA is used to store configurations which are rarely changed.

- **Dynamic ARA**
  A dynamic ARA allows you to change the configuration in the database without requiring the application to reload all of its configuration information and re-configure itself. This is very useful, especially in a SIP setting. For example, it is very likely that you will need to add SIP users over time. By using a dynamic ARA, you can add these users without reloading the entire system configuration (which would mean that service would be interrupted during this period of time). However, not all Asterisk configurations support dynamic ARA.

2.2.1.3 Web GUI for configuration

There are many web-based GUIs for configuring and managing Asterisk. Using a GUI is very convenient for a user to manage Asterisk, as the user does not need to manually change configuration files or manually reload Asterisk. In most cases, the user can simply click to configure items, then click to apply the changes.

[‡] You can even use Open Database Connectivity (ODBC) to store configurations in databases which support ODBC.

2.2.2 Call Detail Records (CDRs)

Call Detail Records store information about calls which were processed by Asterisk. CDRs can be stored in databases. Subsequently an operator of the IP-PBX can use these CDRs to generate billing information. There are many fields in a CDR (for a complete list see [21]), the following are the most important fields for billing:

accountcode	an unique string (up to 20 characters) identifies a subscriber who should pay for this call
src	the souce extension. Because we support users to place calls from more than one device, so users can know where this call is placed.
dst	the destination extension. By checking the destination extension, we can compute the rate for this call
billsec	the call duration from answer to hang up (time is measured in units of seconds)

2.2.3 Asterisk Gateway Interface

The Asterisk Gateway Interface (AGI) is a programming interface for using programming languages in a dialplan. By using AGI, you can do many tasks. Most popular programming languages (such as PHP, Perl, and Python) can be used. This feature is very useful, as you can control Asterisk or communicate with other systems via a network or even a database in a dialplan. Example 4 shows what an AGI script might look like.

```
agi_request: test.py
agi_channel: Zap/1-1
agi_language: en
agi_callerid: 0707100100
agi_context: default
agi_extension: 123
agi_priority: 1
```

Example 4. Example of AGI script

An AGI script is composed of a list of variables and values. The first variable and value defines which script should be run when the dialplan call this AGI script. The rest define what environment should be used when the script is run. Therefore, you can use a general programming language to do advanced logic, manipulate a database, or communicate to another server via the Internet. Using such scripts is more flexible and powerful than using a static dialplan. However, using a program means that the time to process a dialing event is no longer a fixed delay, but can be quite variable depending on the complexity and run-time behavior of the program that is executed.

2.3 Java EE

Java Platform, Enterprise Edition (Java EE) builds on the solid foundation of Java Platform, Standard Edition (J2SE) and is the industry standard for implementing enterprise-class service-oriented architecture (SOA) and next-generation web applications.[22] The Java EE platform provides a number of services beyond the component hosting of Servlets, Java Server Pages (JSPs), and Enterprise JavaBeans (EJBs). Fundamental services include support for XML, web services, transactions, and security.[23] Since Java EE is based on the Java platform, you can use any Application programming Interfaces (APIs) which belong to J2SE. The GUI for these applications could be created using Swing or Abstract Window Toolkit (AWT) or by a HTML-based GUI.

By using Java EE, the applications can utilize the underlying platform to provide security, database manipulation, and transaction control. There are additional advantages of using Java EE. Your Java EE application can run in any Java EE application server which supports the Java EE standard. That means you are not tied to a specific vendor of a Java EE application server. According to http://java.sun.com/javaee/overview/compatibility.jsp there are 13 Java EE 5 compatible implementations from a total of 12 organizations. Another advantage is that you can easily reuse components which are implemented following the Java EE standard.

Security is a very important aspect of enterprise applications. Security for accessing resources is facilitated by using Java EE. The security model is based on roles. A programmer can define what roles can access what resources. This security model is already implemented by the application server. You only need to change the configuration files of the application server to define roles and users. Java EE also supports HTTPS to secure HTTP traffic. You can use a self-signed certificate or a certificate signed by a trusted certificate authority. As to the security of SIP and RTP, the Opticall solution uses Virtual Private Networks (VPNs) to carry this traffic. We will assume that since all of the traffic will be encrypted that we do not have to further consider the security of SIP and RTP. Another reason for using VPNs in the Opticall solution is to handle the problems caused when signaling and media having to pass through multiple Network Address Translations (NATs). Fortunately, using VPNs solves both the NAT traversal problem and provides security for the SIP and RTP traffic. (Details of using VPNs for NAT traversal are described in another thesis [24].)

2.3.1 EJB

Enterprise JavaBeans (EJB) technology is the server-side component architecture for the Java Platform, Enterprise Edition (Java EE). EJB technology enables rapid and simplified development of distributed, transactional, secure, and portable applications based on Java technology.[25]

2.3.1.1 EJB components

The EJB components come in 3 types: Session bean, Message-driven bean, and Entities. Each type of EJB component is responsible for a specific task. These tasks may communicate with each other.

Session bean

Session beans are responsible for business and application logic throughout the whole application. They are invoked by EJB clients, such as a web browser or normal GUI based application. As the name suggests, a session bean exists in a session. This session could be between the EJB server and client. After the session, the session bean will be destroyed. Normally session beans are invoked by EJB clients to execute some business logic or to invoke other EJB entities to store persistent state. There are 2 types of session beans: one is stateful and another is stateless. A stateful session bean can remember state for specific a user until the user completes all of his or her actions. In contrast, the stateless session bean cannot maintain state. So the stateless session beans are normally responsible for independent actions.

Message-driven bean

Message-driven beans can also invoke business methods, but message-driven beans are not invoked by an EJB client, instead they are invoked by messages. Thus when a messaging server receives a signal message, a message-driven bean can be invoked.

Entities

EJB entities are responsible for manipulating a database. Instead of using complex Java Database Connectivity (JDBC) code, you simply map your database tables to entities. These entities are actually Java objects. Using these objects it is possible to perform CRUD (creating, reading, updating, and deleting) operations on a database by manipulating the Java objects. In the latest version of EJB, entities no longer belong in the EJB specification. Instead, the Java Persistence APIs (JPA) handles the entities on behalf of the programmer. JPA provides a SQL-like language called Java Persistence Query Language (JPQL) for performing CRUD operations.

2.3.1.2 Advantages of EJB (version) 3

There are a number of advantages to using EJB (version) 3. Specifically we will take advantage of XML and annotations, and metadata-based Dependency Injection. Each of these will be described in more detail below.

Annotation and XML

One of the hallmarks of the EJB component model is the ability for developers to specify the behavior of both enterprise beans and entities declaratively (as opposed to programmatically), using their choice of Java Development Kit (JDK) 5.0 annotations and/or XML descriptors.[26] In EJB 3, a programmer can use both XML and annotation to configure an EJB or an Entity. Using an annotation to configure an EJB is very easy and clear, because the annotation is located in your Java code. However, some programmers still like using XML, because the behavior of an EJB can be changed without changing the code. You can even use both methods at the same time, but XML always has the highest priority. Example 5 shows an example of using XML and Example 6 shows an example of using annotation to declare an EJB.

```
…

<session>
      <ejb-name>HelloWorld</ejb-name>

      <local-home>
            com.opticall.ejb.interface.HelloworldLocalHome
      </local-home>

      <local>
            com.opticall.ejb.interface.HelloWorldLocal
      </local>

      <ejb-class>
            com.opticall.ejb.HelloWorld
      <ejb-class>

      <session-type>Stateless<session-type>

</session>

…
```

Example 5. Example of declaration using XML using annotation

```
…

@Stateless
public class HelloWorld implements HelloWorldLocal{

…

}

…
```

Example 6. Example of declaration

Dependency injection

In the previous versions of EJB, you had to use Java Naming and Directory Interface (JNDI) lookup to access an EJB. Every time you wanted to use an EJB, you needed to write similar code. In EJB 3, JNDI lookups have been replaced by simple configuration using metadata-based Dependency Injection.[27] The difference between these two approaches is shown in the following two examples: the first one uses JNDI to access an instance of an EJB, another is using Dependency Injection to access an instance of an EJB

```
…
InitialContext ctx = new InitialContext();
try {
HelloWorldBeanLocal helloWorld = (HelloWorldBeanLoca)
ctx.lookup("HelloWorldBean/remote");
…
}
…
```

Example 7. Example of using JNDI

```
@EJB
private HelloWorldBeanLocal helloWorldBean;
```

Example 8. Example of using DI

2.3.2 JBoss

JBoss Application Server (JBoss) is an open source implementation of the Java EE suite of services. It is comprised of a set of bundles for enterprise customers who are looking for preconfigured profiles of JBoss Enterprise Middleware components that have been tested and certified together to provide an integrated experience.[28]

The JBoss runs on a Java virtual machine (JVM). On top of JBoss are your enterprise applications, as shown in Figure 3.

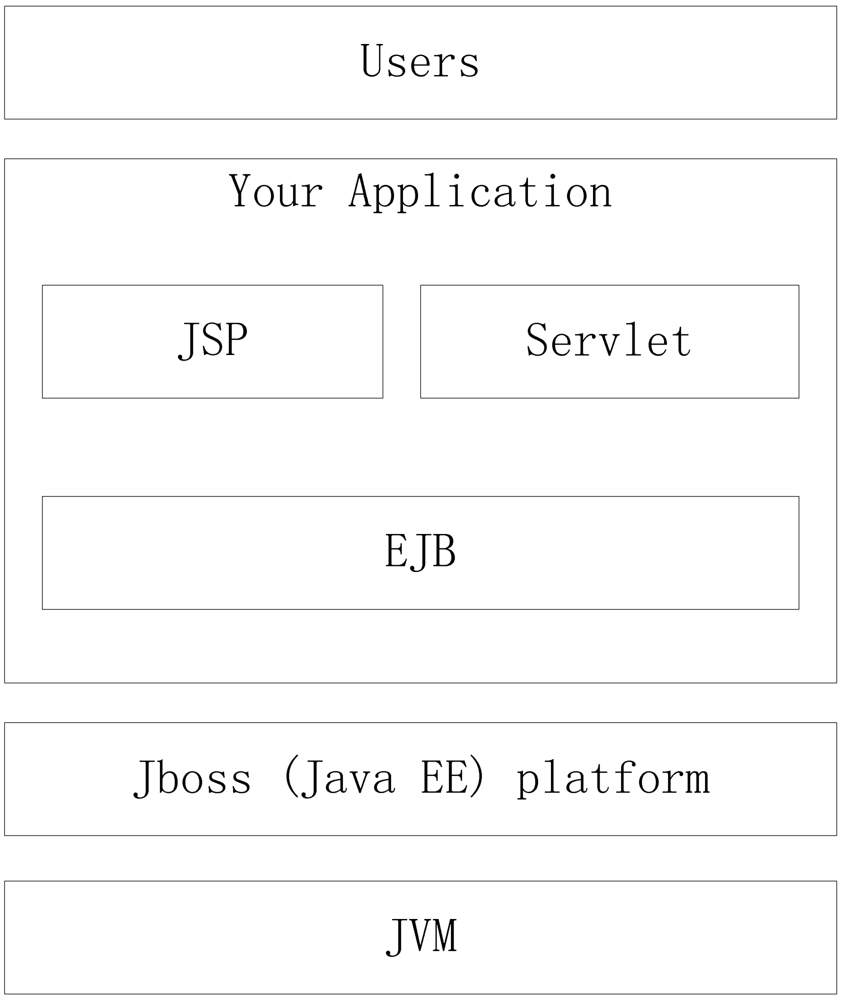


Figure 3. Web application structure in JBoss

2.3.2.1 Authentication of JBoss

JBoss provides a feature for authenticating and authorizing users. The feature is called a security domain. A security domain defines different roles and different resources. Each user has one or multiple roles. Each resource can be accessed by one or multiple roles. We define this security domain in the file "login-config.xml". In this file, you specify how to authenticate users. You can authenticate users based upon a user name and password stored in a file, using a relational database and hashed password, or even using a LDAP server. In a security domain, if you use a relational database, then you can also define an SQL sentence to be used by JBoss to determine whether the password is correct and what role the user has. After defining the security domain, the programmer has to define which resources should be accessed by which roles. By using a security domain in JBoss, you do not need to explicitly write any code for authentication or authorization.

Chapter 3 DoD solution

As mentioned earlier, Opticall AB provides a solution called Dial over Data (DoD), which can provide a low cost and flexible solution for placing a VoIP call. Opticall currently use SIP as the protocol to establish VoIP call sessions. By using DoD, users can place calls from a computer, mobile phone, or fixed phone -- while taking advantage of VoIP when possible. No matter whether you have a SIP user agent or not, you can place low cost calls. In practice the typical user has no idea that they are placing a VoIP call, they simply want to place a call. One of the goals of DoD is that it should allow the user to simply place the call, while DoD hides the details of doing this at low cost. Unfortunately, this was not the case of the existing solution - hence the need for at least one of the improvements presented in the next chapter of this thesis. However, before we present the problem and its solution, we need to introduce the current DoD solution in this chapter.

The DoD solution includes three components: (optional) Symbian clients, DoD server, and IP-PBX (Asterisk). The Symbian client is used to help a user to place a call via DoD from a mobile phone. The DoD server is used to process requests from both users (using computer) and Symbian clients, and present web interfaces to users. The IP-PBX is used to route outgoing calls and receive incoming calls. The IP-PBX is assumed to be connected to an upstream VoIP provider who is connected to one or more PLMNs and PSTN, so the calls from/to IP-PBX will go through this VoIP provider. This chapter will explain these three components in detail.

3.1 Two methods to place VoIP calls

From the user's point of view, the user simply wants to place and call and do so at low cost. The user does not care about how this is accomplished (provided it does not take significant extra effort). DoD solution providers two primary methods to place such calls: “**call through**” and “**call back**”.

3.1.1 Call through

We assume that a user wants to use a mobile phone of Symbian platform to place a call. After the user dials the destination number and presses the green (“Call”) button, the Symbian client will detect this action. The Symbian client stops the normal processing of this action at the operation system level. Instead of dialing this number and placing a call via the PLMN, the Symbian client causes the mobile phone to dial the access number of our IP-PBX, see Figure 4, step 1. The Symbian client makes step is transparent for users, so users can continue using their traditional method of dialing to place calls. The IP-PBX validates the incoming number of user A and answers the incoming call. At this point the Symbian client sends Dual-Tone Multi-Frequency (DTMF) tones to the IP-PBX to tell it the destination number (user B’s number), see Figure 4, step 2. Finally, the IP-PBX places a call to user B. After B answers this call, the IP-PBX bridges these two calls, see Figure 4, step 3.

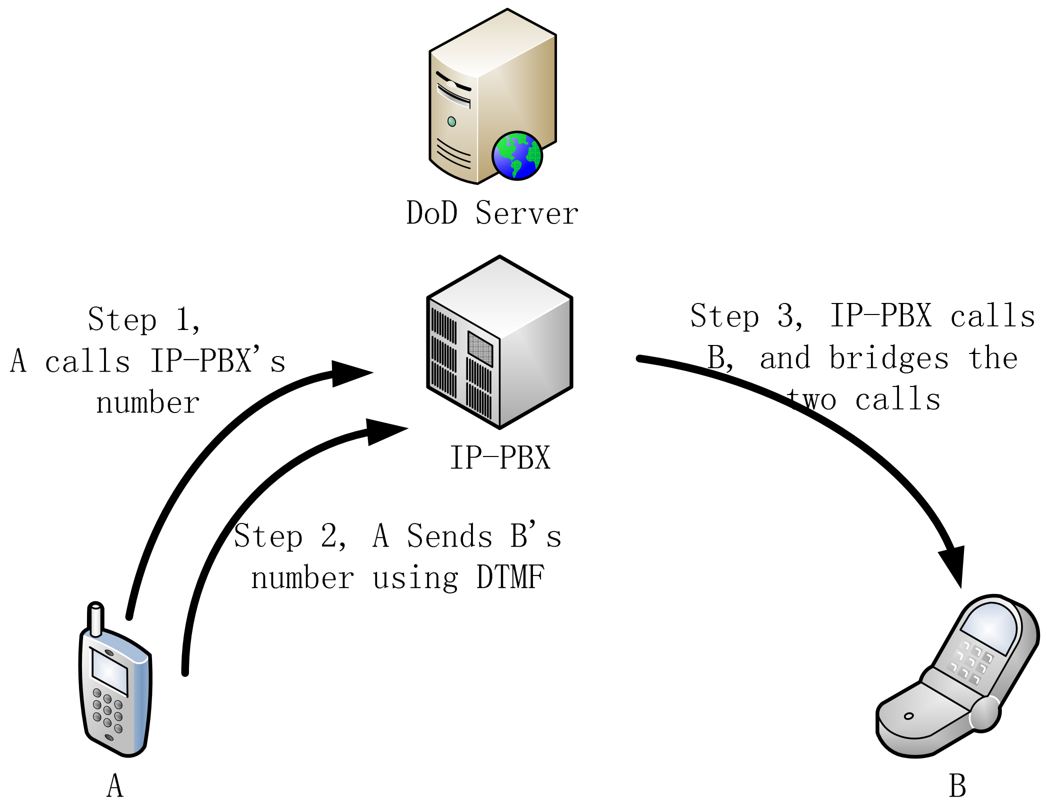

DoD Server
Step 1, A calls IP-PBX's number
IP-PBX
Step 3, IP-PBX calls B, and bridges the two calls
Step 2, A Sends B's number using DTMF
A
B

Figure 4. Call through procedure

The Symbian client is not essential to using call through. Users can manually dial the access number of the IP-PBX. After the IP-PBX validates the user and answer the incoming call, the user can manually dial the number of the destination. Thus the user can use any brand of mobile phone (including those which are too old to install new software). This same method can be used if the user wants to place calls from fixed phones. However, the weak point of placing a call through calls without a Symbian client is that users have to remember two numbers (the access number of the IP-PBX and the number of destination) when dialing and they must explicitly dial the access number of the IP-PBX first and wait until this call is answered before dialing the actual number of the party that they wish to reach. We can see that the call through method of placing a call is essentially the same as the widely used approach to making low cost calls through a local access number.

By using **call through** to place a call, the user has to place a call to the IP-PBX which will normally be in the same country as caller (i.e., this will ideally be a local call). Subsequently the IP-PBX has to place another call to the callee, thus the cost is the sum of the cost of these two calls. The sum of these two calls can be lower than the cost of a direct call in several cases. For example, if the IP-PBX is within the local calling area of the user, then the call to the IP-PBX might be very low cost (for example, in some calling plans this might only be a fixed opening charge and no per minute charges, flat rate, or some other charging scheme). The call from the IP-PBX to the callee can take advantage of the IP-PBX's connection to a SIP trunk, a multiplexed leased trunk, the volume calling prices of the IP-PBX, etc. If a distributed IP-PBX is used (for example in the case of a multinational firm), then the traffic between the IP-PBXs can be sent cheaply and the call connected to the callee might be a local call.

However, there are also a few problems about call through. The first concerns sending DTMF tones. DTMF tones are set over a cellular network as audio tones – when received at the IP-PBX side, the IP-PBX must decode these audio tones. However, the DTMF tones can be lost or distorted by the cellular network. The result is that the correct call might not be set up - unless there is some means to improve the reliability of signaling the callee's number to the IP-PBX. Another problem is validating user. Currently the IP-PBX validates the user based upon the incoming caller ID. This feasible because the IP-PBX already knows all the numbers of all of the valid users. Hence it compares these numbers to the incoming caller ID. If the incoming caller ID matches one of these numbers, then this user has the right to continue to place a call. However, some mobile operators do not transfer the caller ID to the called destination or there is an extra charge for this service. An example of this problem is Companhia de Telecomunicações de Macau S.A.R.L. (CTM)[29] which is a mobile operator in Macau. If the IP-PBX does not get a caller ID from the caller, then this user will be treat as an invalid user and the call through attempt will be rejected. An additional problem occurs because the caller ID of the caller can be faked, thus there is a risk that the IP-PBX can be fooled into making calls for an illegitimate caller.

The traditional solution to the problem of not having the caller ID or a faked caller ID is to force the caller to enter their account number and a password to place the call through call. While this approach is widely used in practice it requires a lot of extra work by the caller. This leads to an alternative way of verifying the legitimacy of the caller - call back. This method will be described in the next section.

3.1.2 Call back

While call through is often perfect for a caller inside the country where the IP-PBX is. If a caller is abroad or he/she cannot send his/her caller ID to the IP-PBX, then the cost of this call is still high or perhaps the caller cannot place a call through call because of a roaming limit by the mobile operator. The second way to place a call in DoD solution is called “Call back”. Consider about two persons A and B. User A is abroad and wants to call B. First user A sends a HTTP request to one of Opticall’s DoD servers by using a Symbian client or a web browser, see Figure 5, step 1. This HTTP request includes A’s number and B’s number. The DoD server receives this request, then causes the IP-PBX to place two calls, see Figure 5, step 2. The first call is to A. After A answers this call, then the IP-PBX places a call to user B, see Figure 5, steps 3a and 3b). After A and B both answer their calls, then the IP-PBX will bridge these two calls. The scaling is roughly the same as the call through approach, but has the added delay of communicating the HTTP request. However, the call set up for both calls could be done in parallel; while in the call through approach the call setups occur sequentially, thus their call set up delays are additive.

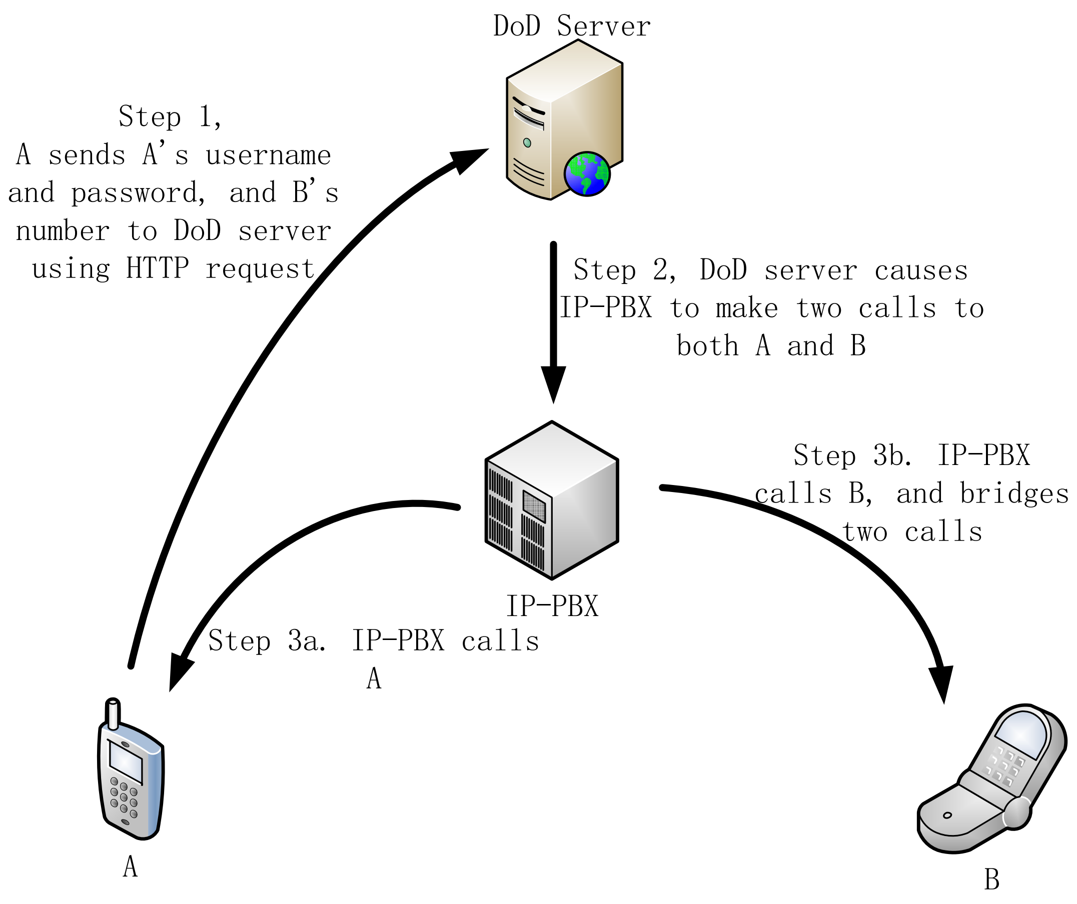

DoD Server
Step 1, A sends A's username and password, and B's number to DoD server using HTTP request
Step 2, DoD server causes IP-PBX to make two calls to both A and B
Step 3b. IP-PBX calls B, and bridges two calls
IP-PBX
Step 3a. IP-PBX calls A
A
B

Figure 5. Call back procedure

There are many ways to tell the IP-PBX A's number and B's number. Using a Web GUI (via the DoD web server), the user can enter A's number and B's number into a web form and presses submit. The web GUI sends a POST request, this causes the IP-PBX to place the call back calls. In addition to the web GUI, we could also use Short Messaging Service (SMS) or DTMF to tell the IP-PBX the phone numbers, in order to cause the IP-PBX to place the call back calls. An advantage of using HTTP/HTTPS is that (1) the HTTP traffic could use Transport Layer Security (TLS) (thus allowing certificate based authentication of both A and the IP-PBX) and (2) the IP-PBX is calling the caller - hence the IP-PBX can verify that this is a legitimate caller and that this caller (caller A) is allowed to call B - before placing either call.

3.1.3 Other call methods

3.1.3.1 Using SIP user agent

Since we use Asterisk as our IP-PBX, you can connect your SIP user agent to the IP-PBX. As a result a user can use their SIP user agent to place outgoing calls to PSTN or PLMNs via this IP-PBX. Of course the user can also use their SIP user agent to place calls to other SIP user agents. Each user in the DoD solution will automatically have a SIP account with the IP-PBX acting as their SIP proxy. Users will be charged they place an outgoing call via the IP-PBX. Users can modify their SIP account by using web GUIs at their DoD server. The modified SIP account is connected to user account in the DoD solution, because we save the username in the field "userfield" in each CDR. In addition, users can place a call to someone's SIP account via the DoD server using call through or call back. For instance, when using call through, user A wants to call the SIP phone of user B from their mobile phone. Instead of entering the number of B, user A enters the SIP account of user B (see figure 4, step 2). Then the IP-PBX will place a call to the SIP phone of user B. A similar approach can be used when using call back. User A sends a HTTP request including the SIP account of user B (see figure 5, step 1), then the IP-PBX will place a call back call between user A and the SIP phone of user B. Currently, there is a limitation when using call through to call someone's SIP account from a cellular or fixed phone: when the user wants to enter B's SIP account, after establishing a call to the IP-PBX -- as the user can only enter digits, thus the SIP account you want to call must consist only of digits.

3.1.3.2 Incoming calls

The users in DoD solution not only place calls, but also can receive calls. Each user will be assigned a number which can be reached from the PSTN and PLMNs. If anyone wants to call user A, they can call this number which was assigned belongs to user A. Subsequently user A's local extension will receive this incoming call. If user A enables his/her followme feature (which will be explained in next paragraph), then the defined followme number will receive this call.

3.1.3.3 Followme call

Followme is a feature which enables callers to reach you anywhere. Assume that user A has a local extension (in this case a SIP user agent), a mobile phone (0707100100), and a number (0757008811) assigned by the IP-PBX. Using the later number this user can be reached from the PSTN or PLMNs via the IP-PBX.

Situation 1: User A has enabled his/her followme feature (set to his/her mobile phone: 0707100100). When someone calls 0757008811 from the PSTN or PLMNs, both his/her local extension and this followme number will ring together. If he/she is in his/her office, he/she can pick up his/her local extension. If he/she is outside office, for example when traveling, he/she can answer his/her mobile phone. No matter which phone he/she picks up, both will stop ringing.

Situation 2: If user B places a call back call or call through call to reach user A's local extension and user A has enabled the followme feature, then both user A's local extension and followme number will begin ringing. Just as in situation 1, he/she can pick up whichever phone he/she wants to use.

Followme feature is designed for someone who often needs to travel or has multiple work places. Users can enable this feature and set their followme number as their mobile number when the user is traveling. Similarly the user can set their followme number as their home number when they are working at home in order to receive incoming calls on their fixed telephone.

3.2 Caller ID

In order to show the correct caller IDs to different devices, we change the caller IDs to the user's assigned extension before the IP-PBX places the call to user B. If user A uses the DoD solution to place a call to a mobile or fixed phone, the number that will be visible to the callee is the number that user A was assigned by the IP-PBX. There many different situations in which the DoD solution should display a different caller ID than the actual caller ID of the user. The following subsections (and figures) will explain when and how we display these different caller IDs. In all of these examples, we assume user A has mobile number 0707100100 and a number assigned by IP-PBX (which is 0757008811). The access number of the IP-PBX is 0757008810. User A will use this access number to place a call through call. User A's local extension is 8811. We will also assume that user A has set his/her followme number to 0707200200.

3.2.1 Caller ID when using Call through

When user A places a call using call through to a mobile number, the caller ID (CID) presented at the callee's mobile should be the number which is assigned by IP-PBX. See figure 6. As a result the callee will always see only a number that can be used by the callee to call user A via the IP-PBX. This makes the number (or SIP URI) used to call the callee invisible to the callee.

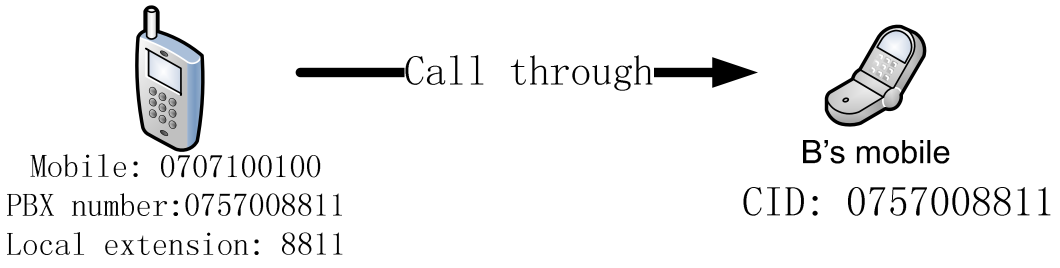


Figure 6. Showing the user's IP-PBX assigned caller ID when using call through

When user A places a call through call to another user of the same DoD system, in this case user A simply called user B's local extension, then user B's device should display the caller ID as user A's local extension. If user B has enabled their followme feature, then the followme phone should display the caller ID as user A's IP-PBX number. See figure 7.

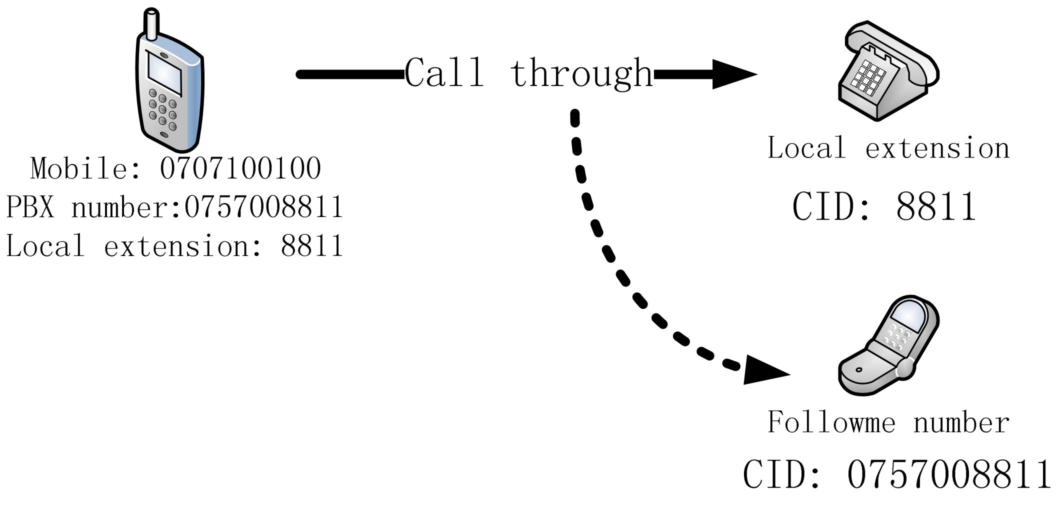


Figure 7. Showing different caller IDs when using call through between two users' connected to the same DoD system

3.2.2 Caller ID when using Call back

When user A places a call back call to someone’s mobile number, then user A will first receive a call with the caller ID of user A’s IP-PBX number, because user A initiated this call. After user A answers this call, then the callee will receive a call with the same caller ID. See figure 8.

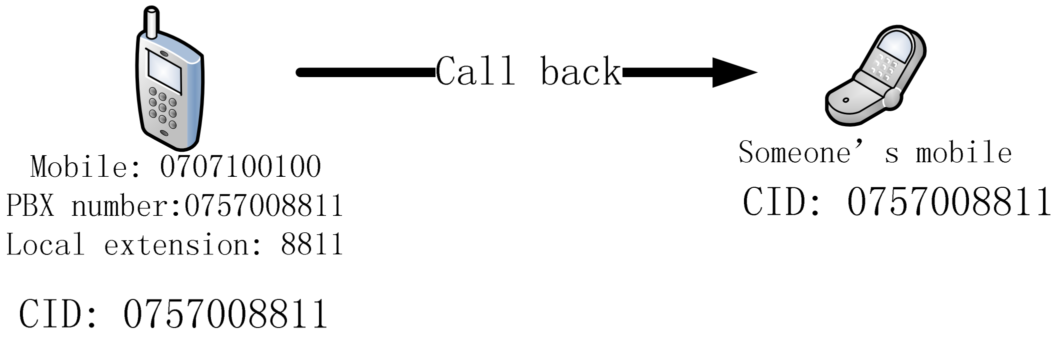


Figure 8. Showing caller ID when using call back

If user A places a call back call to user B’s local extension (this assumes that both users are in the same DoD system). The local extension of user B will receive a call with the caller ID of user A’s local extension. If user B has enabled their followme feature, the followme phone will receive a call indicating as the caller ID user A’s IP-PBX number. See figure 9.

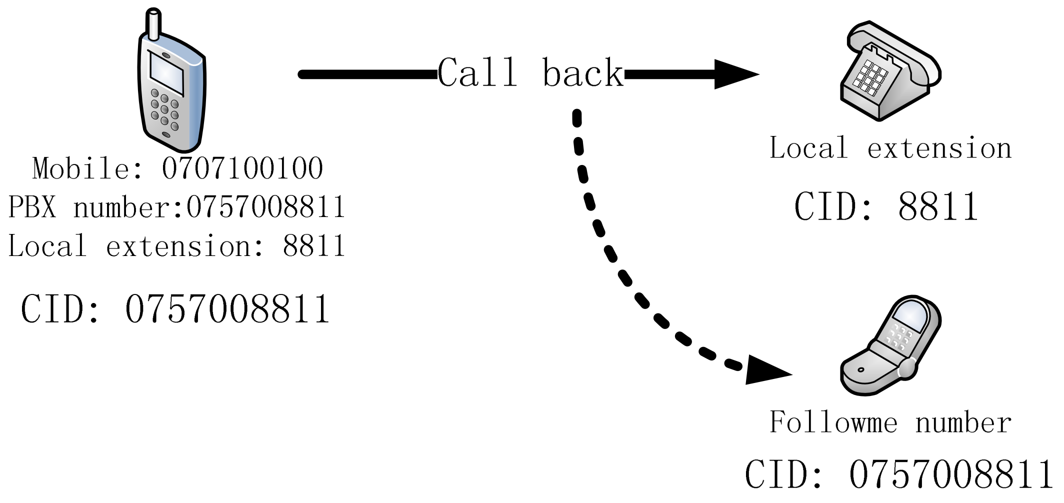


Figure 9. Showing caller ID when using call back

As can be seen in both the call through and call back case, the called user sees a consistent call ID (i.e., it is always the number that they can use to call the caller back). Additionally, in the case of calls sent via public telephony networks the caller IDs are all valid caller IDs for the party making the call to use. However, there is a clear difference between this behavior and the behavior that a caller and callee who have the same mobile operator and are members of a closed calling group would see, in this case the caller and callee would both use and see only a local extension number.

3.2.3 Caller ID when using SIP call

Users also can use a SIP user agent to directly place calls. If user A places a call using their SIP user agent to someone's mobile phone, then the callee's mobile will receive a call with a caller ID being the number assigned by the IP-PBX to user A. See figure 10.

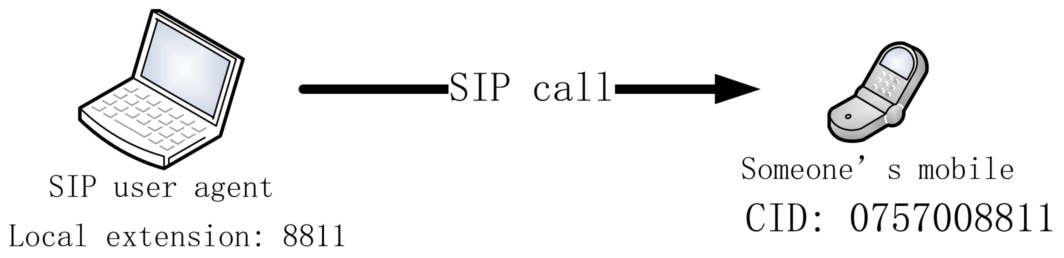


Figure 10. Showing caller ID when using SIP user agent

If user A places a call using their SIP user agent to user B's local extension, the caller ID presented at user B's local extension should be the local extension of user A. If user B has enabled their followme feature, then the followme phone will receive a call showing as the caller ID the number assigned by the IP-PBX to user A. See figure 11.

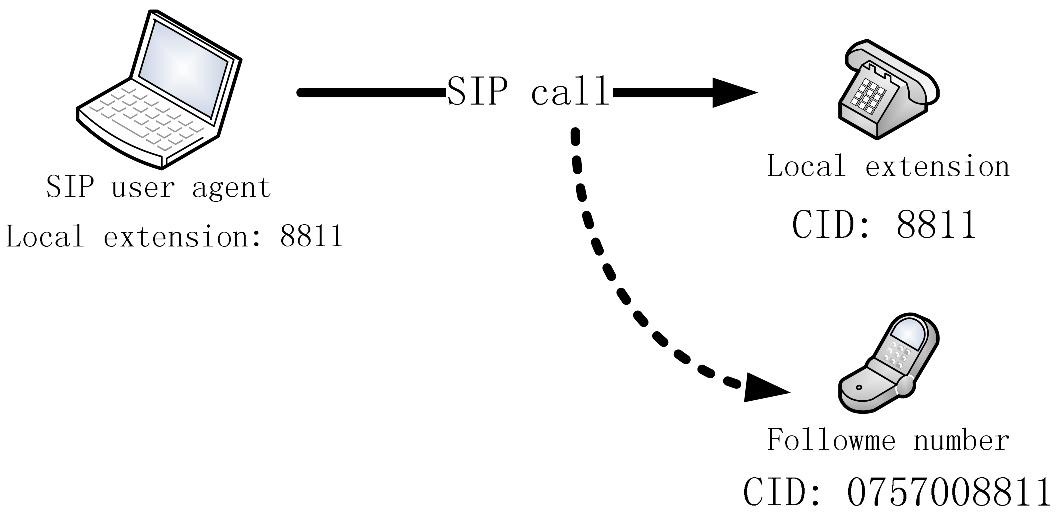


Figure 11. Showing caller ID when using SIP user agent

3.2.4 Caller ID when receiving Incoming call

When someone calls user A's IP-PBX number, the local extension of user A will receive a call showing as the caller ID the real number of the caller. If user A has enabled their followme feature, then the followme phone will receive a call also showing as the caller ID the read number of the caller. See figure 12.

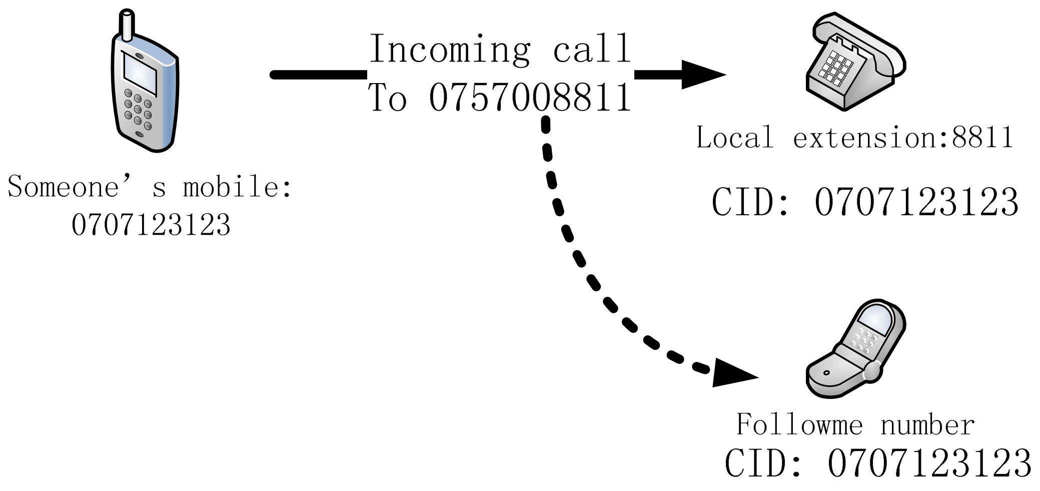


Figure 12. Showing caller ID when receiving a incoming call

Ideally if the caller is also another user of this same DoD system, then the caller ID of the incoming call as displayed at the local extension should be the caller's local extension number. This would have the advantage that all calls between users of the same DoD would all appear on local extensions as calls both to and from local extensions. Additionally, it would mean that a caller could always dial the IP-PBX number of the callee, but this would be handled as a call between local extensions when both parties were using their local extensions. The advantage of this approach is that the caller does not have to know or even think about if the callee is at their local extension or not - they simply always dial the same number for this party. Note that the reverse is not true, i.e., the caller cannot always call the callee at their local extension -- unless the caller is using the Symbian client or their SIP UA (i.e., this would not work with a mobile or fixed phone - as the switched attached to the access network would not be able to interpret these "local extension" numbers - since they do not exist in the actual local access network). However, in some mobile and fixed phone networks there is a facility to define short numbers, if there were a short number defined for each of the relevant "local extension" numbers, then a similar behavior could be provided; but this would not work with all mobile and fixed phone networks.

3.3 Three components of the DoD solution

3.3.1 Symbian client

Java is platform independent language, thus software using J2ME can run on almost all mobile devices equipped with Java. However, J2ME (expect on the Android platform) cannot detect and control the operating system in a comparable manner as a Symbian application can. Thus J2ME cannot detect an outgoing call, send DTMF, and terminate a call. Using J2ME is possible to implement a client for call back and call through, but the user cannot use their traditional method of placing calls, but instead explicitly launch Java client first, then enter the necessary numbers, and finally choose which method (call back or call through) to use. This requirement of starting a Java application is viewed as an impediment to the routine use of the DoD solution, hence the need for a Symban client.

The Symbian client is software running on top of a Symbian operating system platform. This software is an optional component in the DoD solution, but it helps users to place calls. The Symbian client can be used to place call back calls and/or call through calls. This software is written in Symbian C++, which is a system level programming language of the Symbian platform. Using this client, user can place calls in the traditional way (i.e., by simply entering a number and pressing the green dial/call button). The Symbian client detects this event and can process the call using the calling method selected by the user. Similar clients could be implemented for other platforms, for example, Android, iPhone OS, Windows Mobile, and Linux.

Additionally, users may want to place call back or call through calls without a Symbian client (for example from non-Symbian mobile phones and fixed phones). For call through, user must manually dial the access number of IP-PBX, then dial the callee’s number. For call back, on a phone equipped with a web browser the user can browse to a web page for the DoD server, then enter numbers and click “submit” to place a call back call.

3.3.2 IP-PBX (Asterisk)

The IP-PBX is used to place calls via a SIP trunk, process HTTP requests from clients, and validate users. We assumed that the IP-PBX is connected to a SIP trunk which is connected to PSTN and PLMNs, so that this IP-PBX can place calls to PSTN or PLMNs and also receive calls from PSTN or PLMNs.

When user places a call through call, the IP-PBX will first validate the caller ID of caller. This requires that all of the mobile number of users have already been saved in the database. If the incoming caller ID is the same as one of the numbers associated with a DoD user, the IP-PBX treats the caller as a valid user. After the IP-PBX answers this call, it provides a new dial tone to user. Then user sends DTMF tones using Symbian client or manually to indicate the callee's number. After waiting several seconds to make sure the user is done inputting a number, IP-PBX places an outgoing call to the callee's number[§]. After the callee answers this call, then the IP-PBX bridges these two calls together.

When a user places a call back call, a HTTP (or HTTPS) request which includes the caller's number, the callee's number, the username, and a password is sent to the DoD server. The DoD server creates and sends a call request to Asterisk. After Asterisk receives the request sent from the DoD server, Asterisk places two calls, one to the caller and another one to the callee. After these two calls are both answered, Asterisk bridges these two calls together. If the caller or the callee is not answered, after 20 seconds the call will be terminated.

As mentioned earlier, using a database is an important part in Asterisk. In the IP-PBX databases have been used in several places. Currently three databases are used.

1. CDR

By default, Asterisk generates CDR records as comma-separated text files in the /var/log/asterisk/cdr-csv directory. The file "master.csv" contains all the records. Detail records can be partially configured on a per channel basis, and some of the data for IAX[**] and SIP can be determined on the user level. The Zap configuration for CDR records is determined by the configuration of the ZAP channel in zaptel.conf[30][††]. In order to make it easy for other applications, we save all CDR information in a MySQL database. The following information has been stored in the database: start calling date, durations, source number, destination number, and so on. We also save a username for each record, so we can identify who has placed each call.

2. Dialplan and ARA

The configuration file "extensions.conf" contains the "dial plan" of Asterisk, the master plan for control or execution flow for all of Asterisk's operations. In particular the configuration controls how incoming and outgoing calls are handled and routed. This is where you configure the behavior of all connections through your PBX.[31] In the dialplan, the logic specifies how to route outgoing and incoming calls. One important aspect in the dialplan is how to display caller ID to different devices, as some calls will lead two different kinds of devices ringing at the same time.

[§] The need to wait could be eliminated if a distinguished toke was used to terminate the dialed number, for instance the "#" symbol.

[**] IAX is the Asterisk exchange protocol that can be used for VoIP service within and between Asterisk exchanges.

[††] Zaptel interface cards are used to interface to traditional circuit-switched telephone equipment (both telephones and exchanges).

However, Asterisk only supports one caller ID for each call. After much effort, we found out it is possible use a LOCAL channel to solve this problem. LOCAL is an Asterisk pseudo-channel. Use of this channel simply loops the calls back into the dialplan in a different context. Looping the call back is useful for recursive routing; as it is able to return to the dialplan after call completion.[32] Thus when the IP-PBX is going to place a call to more than one destination, the IP-PBX uses a local channel to place this call. If the destinations of this call are two places, this call will be rerouted to two different context and the caller ID can be suitably modified for each of these calls.

In order to use DoD server to change the dialplan without reloading, we use ARA, and save all the dialplans into a database. (For details about ARA see section 2.2.1.2 on page 9.)

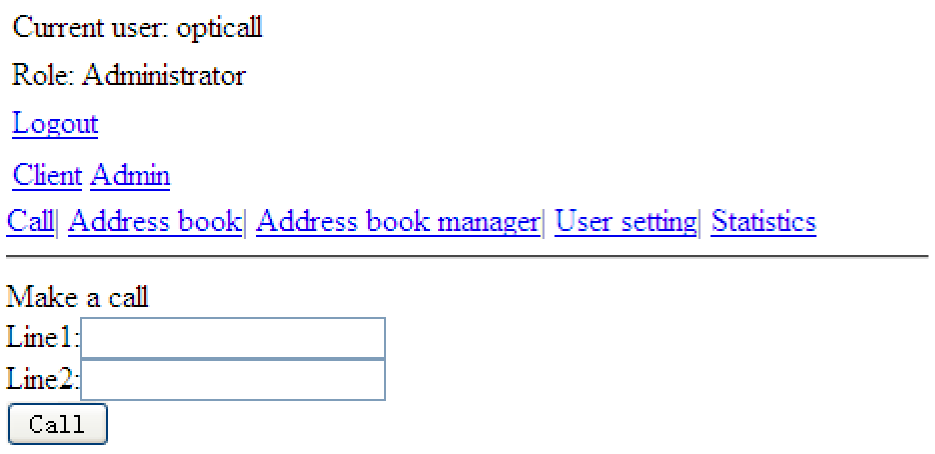
3. Validate user and showing different CID using func_odbc

In order to validate the user, we use information stored in the database. We use the func_odbc in our dialplan. The func_odbc dialplan function allows you to create and use fairly simple dialplan functions that retrieve and use information from databases directly in the dialplan. There are all kinds of ways in which this might be used, such as managing users or allowing sharing of dynamic information within a clustered set of Asterisk machines. When the IP-PBX receives an incoming call, we can use func_odbc to look up whether this caller ID is already in the database as that of a valid user. Similarly if an outgoing call has multiple destinations, we use func_odbc to look up in the database the corrent caller ID to be used for each call.

3.3.3 DoD server

The DoD server provides web GUIs to allow a user to place call back calls and for the administrator to manage both the DoD server and the IP-PBX. DoD server uses made by JavaEE. We use JBoss as the JavaEE container because JBoss is open source software and free to use. The DoD server provides a web GUI for placing call back calls. Additionally the DoD server can place call back calls based upon processing HTTP or HTTPS requests sent by clients. The DoD server also can change some of Asterisk's settings, thus we do not need additional GUI for configuring Asterisk. Using one GUI for both placing calls and configuring Asterisk is an improvement that will be explained in more detail in next cheaper. In addition to calling and management features, the DoD server also provides some extra features, such as a contact book for simplified dialing and a GUI for showing CDR records as a call history.

The following figures are the some of screenshots of DoD server.


Current user: opticall

Role: Administrator

Logout

Client Admin

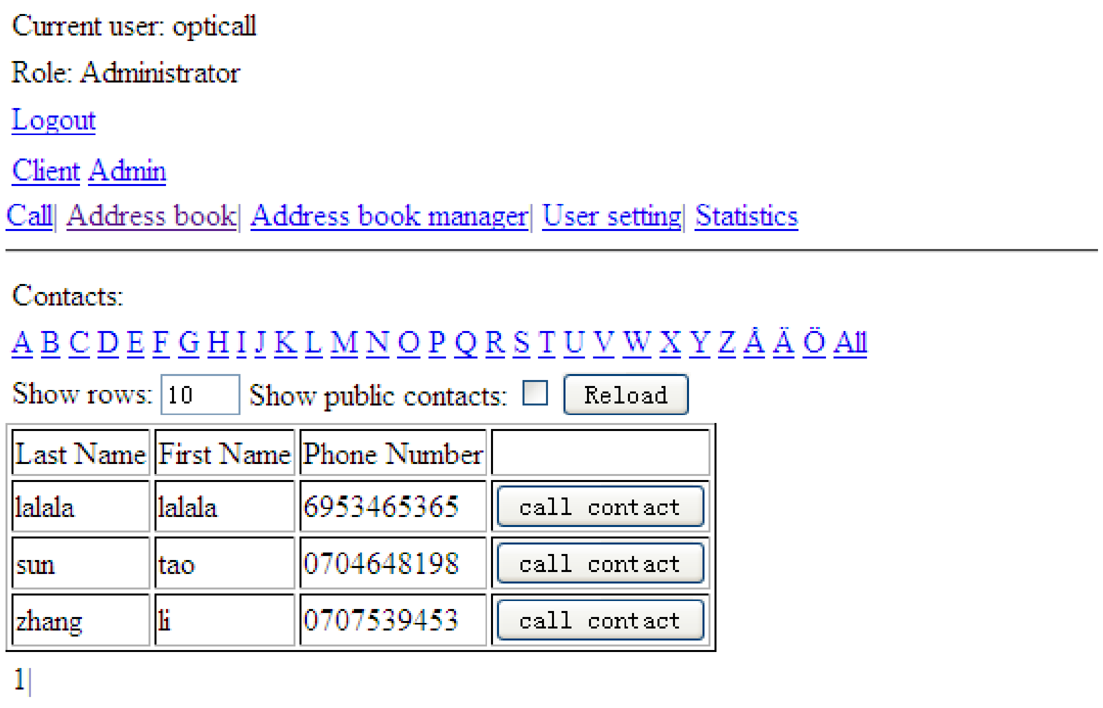
Call| Address book| Address book manager| User setting| Statistics

Make a call

Line1:

Line2:

Figure 13. The web GUI for call back


Current user: opticall

Role: Administrator

Logout

Client Admin

Call| Address book| Address book manager| User setting| Statistics

Contacts:

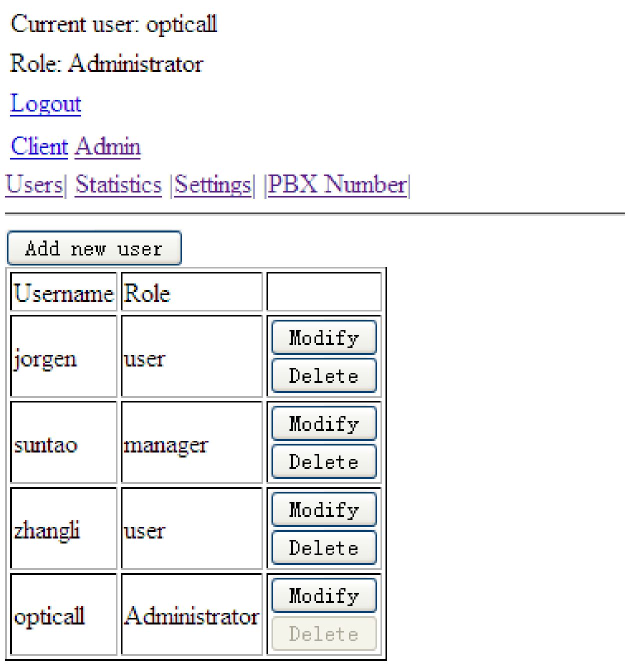
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Å Ä Ö All

Show rows: 10 Show public contacts: ☐

Last Name	First Name	Phone Number	
lalala	lalala	6953465365	call contact
sun	tao	0704648198	call contact
zhang	li	0707539453	call contact

1|

Figure 14. The web GUI for contact book


Current user: opticall

Role: Administrator

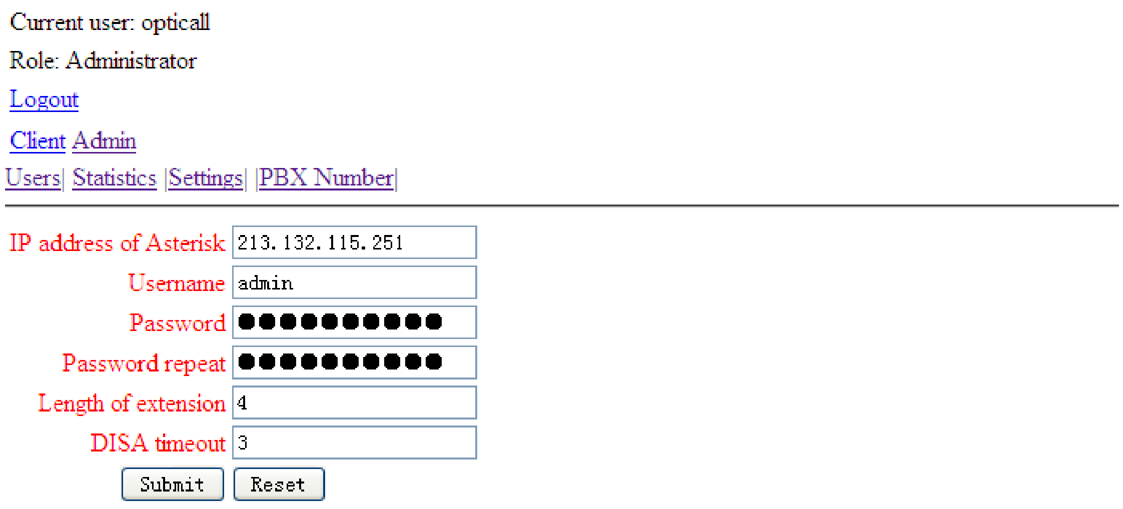
Logout

Client Admin

Users| Statistics |Settings| |PBX Number|

Username	Role	
jorgen	user	Modify Delete
suntao	manager	Modify Delete
zhangli	user	Modify Delete
opticall	Administrator	Modify Delete

Figure 15. The web GUI for administrators to manage users


Current user: opticall

Role: Administrator

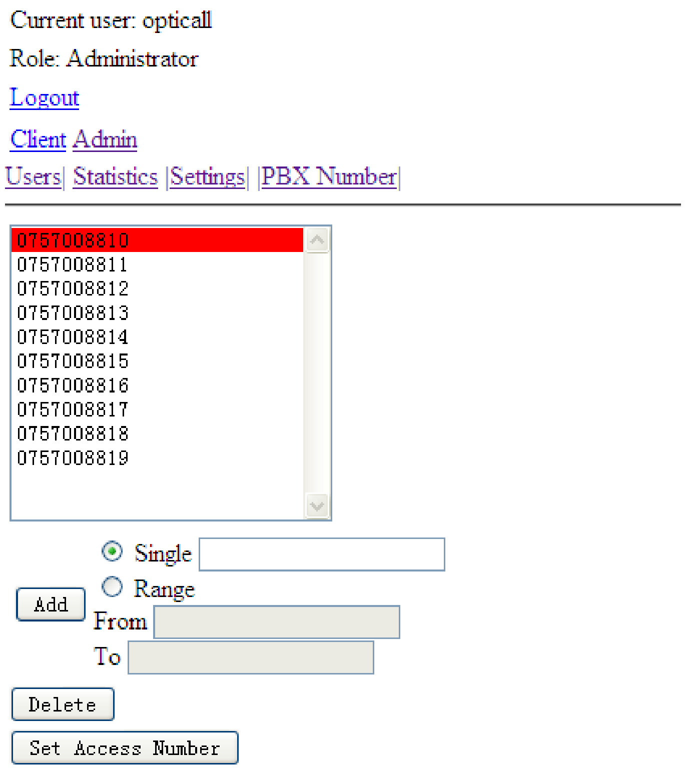
Logout

Client Admin

Users| Statistics |Settings| |PBX Number|

IP address of Asterisk	213.132.115.251
Username	admin
Password	●●●●●●●●●●●
Password repeat	●●●●●●●●●●●
Length of extension	4
DISA timeout	3

Figure 16. The web GUI for administrator to manage the connection between Asterisk and DoD server


Current user: opticall

Role: Administrator

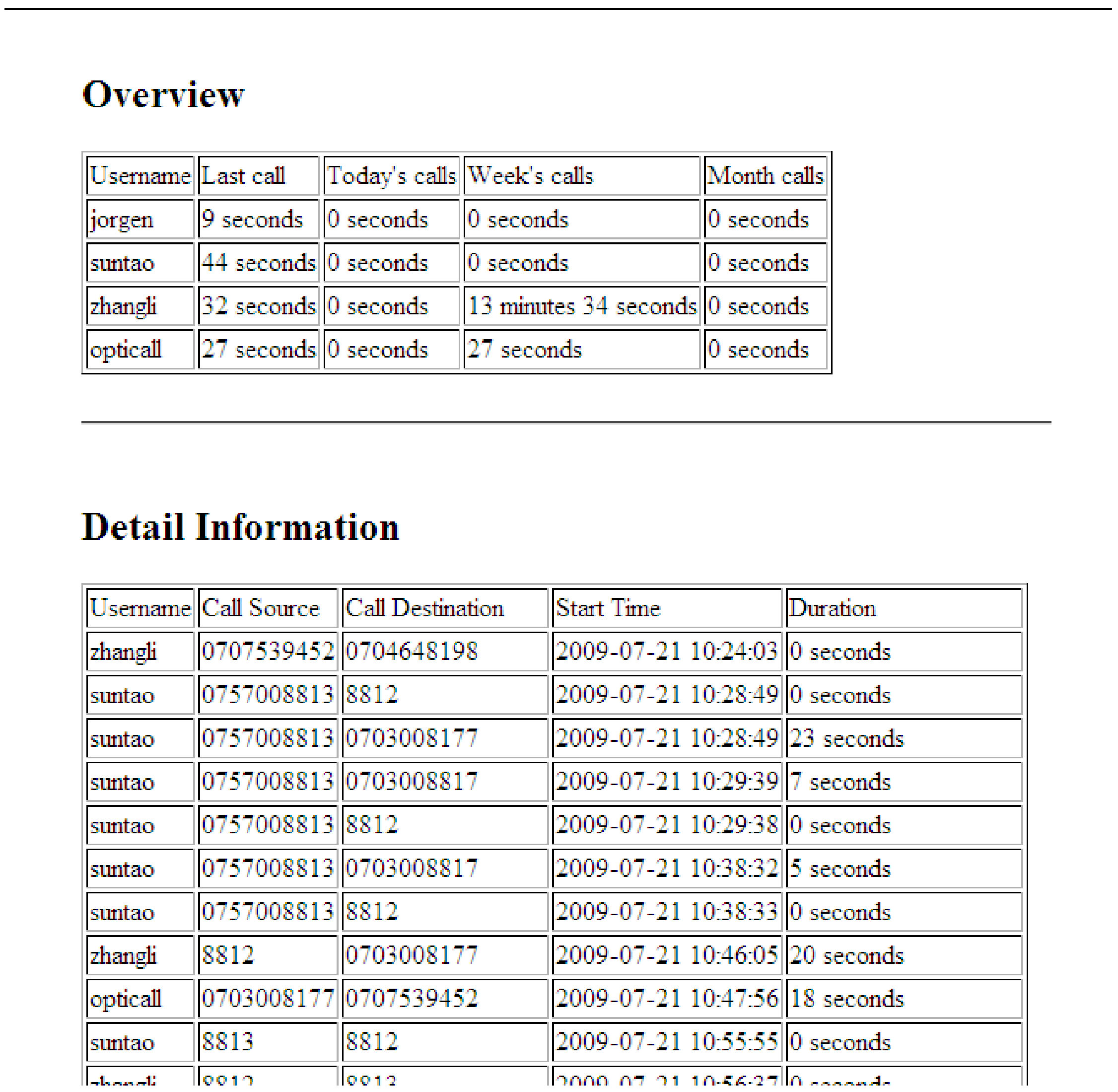
Logout

Client Admin

Users| Statistics |Settings| |PBX Number|

0757008810
0757008811
0757008812
0757008813
0757008814
0757008815
0757008816
0757008817
0757008818
0757008819

Figure 17. The web GUI for administrator to modify the numbers of IP-PBX



Overview

Username	Last call	Today's calls	Week's calls	Month calls
jorgen	9 seconds	0 seconds	0 seconds	0 seconds
suntao	44 seconds	0 seconds	0 seconds	0 seconds
zhangli	32 seconds	0 seconds	13 minutes 34 seconds	0 seconds
opticall	27 seconds	0 seconds	27 seconds	0 seconds

Detail Information

Username	Call Source	Call Destination	Start Time	Duration
zhangli	0707539452	0704648198	2009-07-21 10:24:03	0 seconds
suntao	0757008813	8812	2009-07-21 10:28:49	0 seconds
suntao	0757008813	0703008177	2009-07-21 10:28:49	23 seconds
suntao	0757008813	0703008817	2009-07-21 10:29:39	7 seconds
suntao	0757008813	8812	2009-07-21 10:29:38	0 seconds
suntao	0757008813	0703008817	2009-07-21 10:38:32	5 seconds
suntao	0757008813	8812	2009-07-21 10:38:33	0 seconds
zhangli	8812	0703008177	2009-07-21 10:46:05	20 seconds
opticall	0703008177	0707539452	2009-07-21 10:47:56	18 seconds
suntao	8813	8812	2009-07-21 10:55:55	0 seconds
zhangli	8812	8813	2009-07-21 10:56:37	0 seconds

Figure 18. The web GUI for administrator to show the call history

Chapter 4 Improvements

The goal of this thesis project was to improve the server side of the DoD solution. The existing DoD solution was up and running, but there were a number of problems. For example, the web interface did not show CDR records properly, the caller ID was wrong when a call had multiple destinations, and multiple (inconsistent) web interfaces were need for controlling the complete DoD solution. The following sections will explain each of these problems, and how they were each solved in this thesis project.

4.1 Saving Asterisk configurations in database

By default, Asterisk keeps all configurations as files saving in /etc/asterisk. An administrator can change an Asterisk configuration by modifying these configuration files. However, the Asterisk system has to be reloaded in order for these changes to take effect. For example, to add a new SIP account to Asterisk, you have to modify sip.conf, then reload the SIP settings into Asterisk. In the DoD solution, each user has a SIP account and each user can modify their own SIP account through web GUIs. Additionally when adding or deleting users a SIP account will be created, modified, or deleted. As a result the SIP settings will be changed frequently. Additionally, the dialplan will be changed when users enable their followme feature or following another administrative change. If we have to rewrite the configuration files and reload settings when any settings have been modified, this is both a lot of work and inefficient. Therefore, we decided to use the Asterisk Realtime Architecture (ARA) to solve this problem.

As mentioned at subsection 2.2.1.2 using ARA, the configuration can be saved in a database. ARA also supports dynamic updates of the configuration without requiring reloading. For example, if you want to change a SIP account, you only need to change it in database. Another advantage of this approach is that application can change or manage the Asterisk configuration(s). This can be quite simple, but very power, since the application can change the behaviors of Asterisk simply by changing entries in the database.

In DoD solution, we save SIP and diaplan settings in a database because these two kinds of setting may be changed frequently. In order to save these setting in a database, we need to create database tables for each one. The structure of each table is constant and includes all the information necessary for each kind of setting. After we creating two tables in the database, we can add records to these tables.

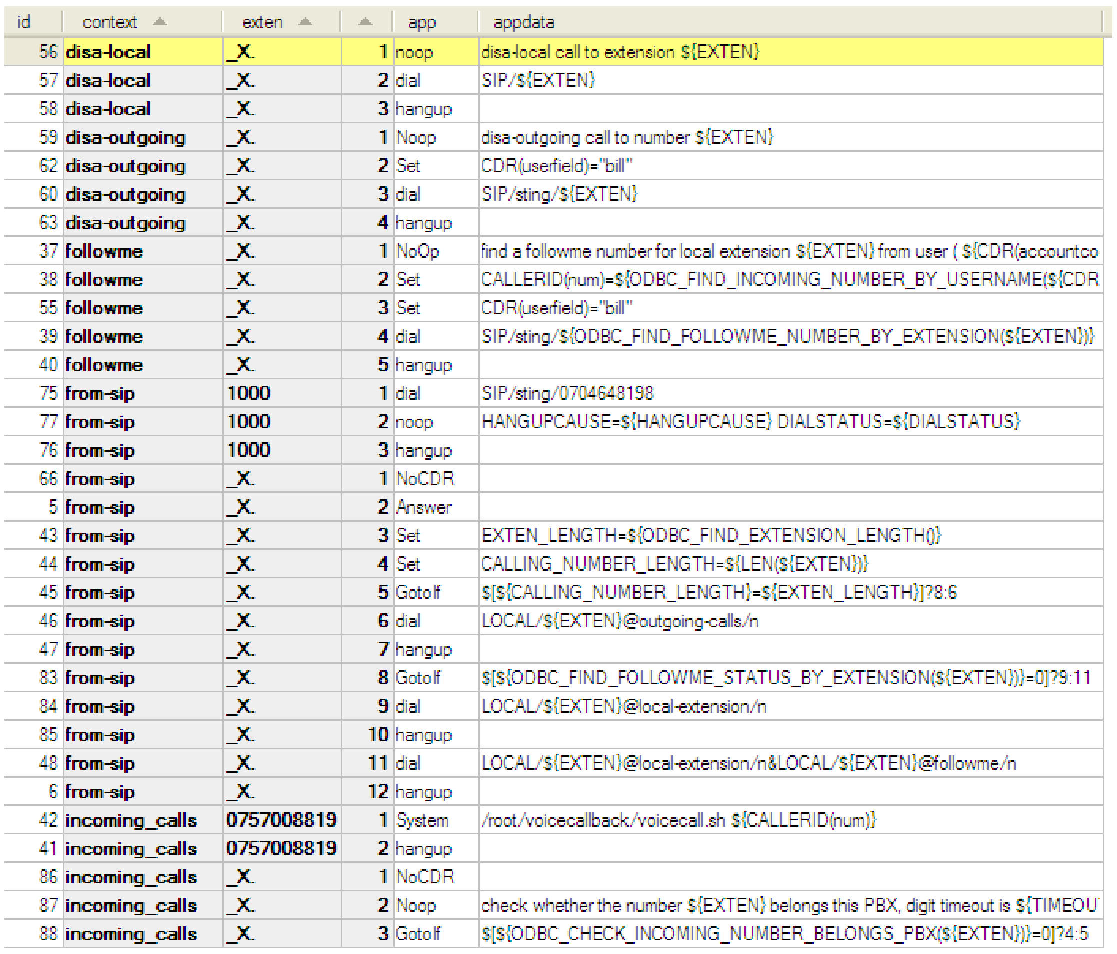
For SIP, each record is a SIP account. When we save the SIP settings in a flat file, we entries of the form “key=value” to define a SIP count, see Example 9. Each key represents an attribute of this account that you can set, each with the value you want to set for this attribute for this account. Use of default values simplifies creating an account, as you only need to specify non-default values. When using ARA, you still use keys and values to define a SIP account, but now the columns are keys and the content of a row in the table contain the associated values. When creating the table, you can specify default values for some (or all) columns. As a result you only need to specify values for those columns that will have a non-default value.

```
[tammari]
type=friend
callerid="Tuomas Tammisalo" <1000>
username=tammari
host=dynamic
secret=********
regcontext=tammari-internal
regexten=1005
dtmfmode=rfc2833
insecure=very
canreinvite=yes
nat=yes
qualify=yes
context=merus-sipphone
pickupgroup=1
callgroup=1
mailbox=1000@default
```

Example 9. SIP account setting in Asterisk

The dialplan could be stored in a flat file (/etc/asterisk/extensions.conf). The dialplan is separated into contexts, with each contexts having one or more exstensions with priorities and commands. This same information can be stored in a dialplan table in the database with each row specifying an extension. Just as with SIP settings, by storing the dialplan in a database there is no need to re-load the dialplan; instead the system will access the relevant parts of this dialplan as it operates.

When using ARA for dialplan, We first create a table for the dialplan in the database. The table structure is fixed and includes all the necessary information for the dialplan. Each record is an extension specifying a priority and command. In order to identify which extensions belong to which context, each record also has a column to save context information, see Figure 19. Although using ARA for dianplan has many advantages such as enabling changesto the dialplan without reloading, easy access for external application, and so on, but there is also a limitation. Figure 19 shows part of a dialplan as stored in the database; while Example 10 shows part of a dial plan stored in a flat file. Example 10 each row is an extension containing an extension number, priority, and a command. Unlike the case of a flat file where the order in the file could be used to specify an implicit priority (and order) , in the cases of storing the dialplan in a database we have to explicitly specify the priority ordering of the records that concern a given extension. Despite this small amount of extra work, we believe that ARA offers many advantages over using a flat file.



id	context	exten		app	appdata
56	disa-local	_X.	1	noop	disa-local call to extension ${EXTEN}
57	disa-local	_X.	2	dial	SIP/${EXTEN}
58	disa-local	_X.	3	hangup	
59	disa-outgoing	_X.	1	Noop	disa-outgoing call to number ${EXTEN}
62	disa-outgoing	_X.	2	Set	CDR(userfield)="bill"
60	disa-outgoing	_X.	3	dial	SIP/sting/${EXTEN}
63	disa-outgoing	_X.	4	hangup	
37	followme	_X.	1	NoOp	find a followme number for local extension ${EXTEN} from user (${CDR(accountco
38	followme	_X.	2	Set	CALLERID(num)=${ODBC_FIND_INCOMING_NUMBER_BY_USERNAME(${CDR
55	followme	_X.	3	Set	CDR(userfield)="bill"
39	followme	_X.	4	dial	SIP/sting/${ODBC_FIND_FOLLOWME_NUMBER_BY_EXTENSION(${EXTEN})}
40	followme	_X.	5	hangup	
75	from-sip	1000	1	dial	SIP/sting/0704648198
77	from-sip	1000	2	noop	HANGUPCAUSE=${HANGUPCAUSE} DIALSTATUS=${DIALSTATUS}
76	from-sip	1000	3	hangup	
66	from-sip	_X.	1	NoCDR	
5	from-sip	_X.	2	Answer	
43	from-sip	_X.	3	Set	EXTEN_LENGTH=${ODBC_FIND_EXTENSION_LENGTH()}
44	from-sip	_X.	4	Set	CALLING_NUMBER_LENGTH=${LEN(${EXTEN})}
45	from-sip	_X.	5	GotoIf	$[${CALLING_NUMBER_LENGTH}=${EXTEN_LENGTH}]?8:6
46	from-sip	_X.	6	dial	LOCAL/${EXTEN}@outgoing-calls/n
47	from-sip	_X.	7	hangup	
83	from-sip	_X.	8	GotoIf	$[${ODBC_FIND_FOLLOWME_STATUS_BY_EXTENSION(${EXTEN})}=0]?9:11
84	from-sip	_X.	9	dial	LOCAL/${EXTEN}@local-extension/n
85	from-sip	_X.	10	hangup	
48	from-sip	_X.	11	dial	LOCAL/${EXTEN}@local-extension/n&LOCAL/${EXTEN}@followme/n
6	from-sip	_X.	12	hangup	
42	incoming_calls	0757008819	1	System	/root/voicecallback/voicecall.sh ${CALLERID(num)}
41	incoming_calls	0757008819	2	hangup	
86	incoming_calls	_X.	1	NoCDR	
87	incoming_calls	_X.	2	Noop	check whether the number ${EXTEN} belongs this PBX, digit timeout is ${TIMEOU
88	incoming_calls	_X.	3	GotoIf	$[${ODBC_CHECK_INCOMING_NUMBER_BELONGS_PBX(${EXTEN})}=0]?4:5

Figure 19. Dialplan stored as a table in a database

```
……
[voice mail]
exten => 123, 1 ,Answer
exten => 123, n ,Playback(tt-weasels)
exten => 123, n ,Voicemail(44)
exten => 123, n ,Hangup

[from-sip]
……
```

Example 10. Dialplan saving in flat file

4.2 Caller ID

The existing DoD server did not show the proper caller ID sometimes, especially when a call had multiple destinations. For example, when user A places a call to a local extension of user B, then the local extension number of user A should be displayed at B’s local extension. However, the number assigned by the IP-PBX to user A was displayed to B.

As mentioned in section 3.2, there are a number of different situations that have to be addressed. Here we define a number of rules that will be used to decide how to display the caller ID.

The first rule is to hide the caller's actual number, instead we want to only display the number assigned to this caller by the IP-PBX.

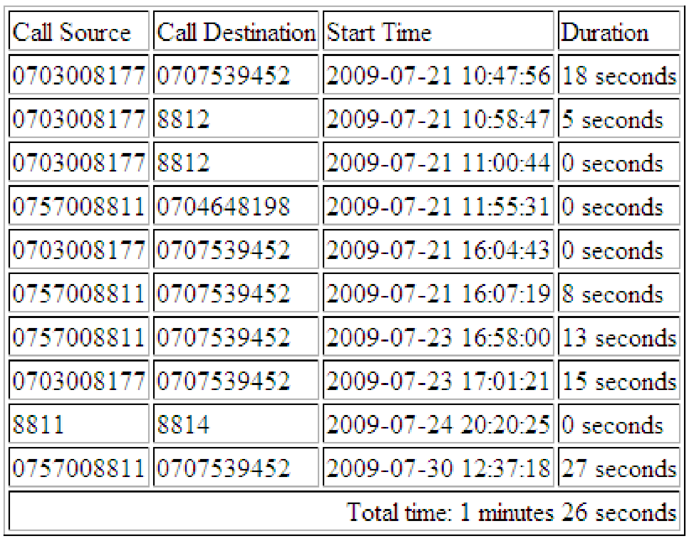
The second rule is that the caller ID should be appropriate for the device where it is displayed. If the destination is a mobile phone, then the caller ID should be the number assigned by IP-PBX as this would be the number that the callee could call to reach this caller. If the destination is a local extension, then the caller ID should be the local extension number -- as this too is the number that the callee could call using this local extension to reach the caller.

We can address both of these rules by using the func_odbc in our dialplan and using local channels to create multiple contexts when there is more than one destination for a call. To implement this we simply need to insert the correct information into the dialplan database table. The procedure for adding or modifying an entry into the dialplan table can automatically update the correct rows to realize our rules.

4.3 Improving CDR records

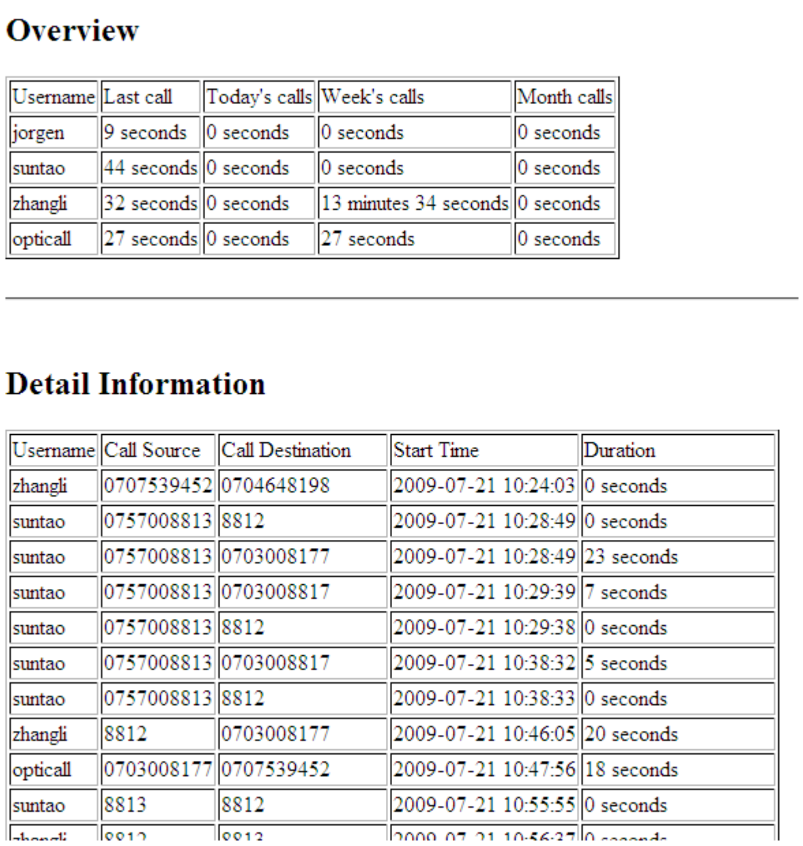
In the existing DoD server, there were several web GUIs for statistical information, for example, showing only the duration of last call, the duration of today's calls, the duration of this week's calls, and the duration of this mouth's calls. This information was not useful to the user. Users may want to know their call history in detail, including call date, destination number, caller number, and so on. In addition an administrator might want to check the call history for one or more users.

In order to provide more information to both users and administrators, we create several new web GUIs to show detail call history. User can get the following information from this GUI (see figure 20): caller's number, callee's number, call time, duration. Since the administrators may need to see all users' call history. We created another GUI for the administrator, see figure 21. Using this GUI, administrators have an overview of all users. If the administrator wants more detail information about a specific user, they can selected this user and see all of this user's call history.



Call Source	Call Destination	Start Time	Duration
0703008177	0707539452	2009-07-21 10:47:56	18 seconds
0703008177	8812	2009-07-21 10:58:47	5 seconds
0703008177	8812	2009-07-21 11:00:44	0 seconds
0757008811	0704648198	2009-07-21 11:55:31	0 seconds
0703008177	0707539452	2009-07-21 16:04:43	0 seconds
0757008811	0707539452	2009-07-21 16:07:19	8 seconds
0757008811	0707539452	2009-07-23 16:58:00	13 seconds
0703008177	0707539452	2009-07-23 17:01:21	15 seconds
8811	8814	2009-07-24 20:20:25	0 seconds
0757008811	0707539452	2009-07-30 12:37:18	27 seconds
			Total time: 1 minutes 26 seconds

Figure 19. Web GUI showing CDR for normal users



Overview

Username	Last call	Today's calls	Week's calls	Month calls
jorgen	9 seconds	0 seconds	0 seconds	0 seconds
suntao	44 seconds	0 seconds	0 seconds	0 seconds
zhangli	32 seconds	0 seconds	13 minutes 34 seconds	0 seconds
opticall	27 seconds	0 seconds	27 seconds	0 seconds

Detail Information

Username	Call Source	Call Destination	Start Time	Duration
zhangli	0707539452	0704648198	2009-07-21 10:24:03	0 seconds
suntao	0757008813	8812	2009-07-21 10:28:49	0 seconds
suntao	0757008813	0703008177	2009-07-21 10:28:49	23 seconds
suntao	0757008813	0703008817	2009-07-21 10:29:39	7 seconds
suntao	0757008813	8812	2009-07-21 10:29:38	0 seconds
suntao	0757008813	0703008817	2009-07-21 10:38:32	5 seconds
suntao	0757008813	8812	2009-07-21 10:38:33	0 seconds
zhangli	8812	0703008177	2009-07-21 10:46:05	20 seconds
opticall	0703008177	0707539452	2009-07-21 10:47:56	18 seconds
suntao	8813	8812	2009-07-21 10:55:55	0 seconds
zhangli	8812	8813	2009-07-21 10:56:37	0 seconds

Figure 20. Web GUI showing CDR for administrator

As mentioned section 4.2, we support different caller IDs at different devices simultaneously by using a local channel. The caller IDs can be correctly manipulated using this local channel, but there are many extra CDR records generated in the database for each call. Because we use a local channel, an extra CDR record is created for each of these local channels. Unfortunately, users are confused by these (useless) extra CDR records; especially when they see that many call happened at almost the same time. These unnecessary CDRs make it hard for the user to find the useful information.

In order to solve this problem, we use the command "NoCDR" for the local channel call in the dialplan. As a result Asterisk will not save the CDR for this call.[33] Alternatively, we could simply filter the CDR entries that are to be presented and show only those that are significant.

4.4 Optimizing code

The DoD server was designed to process HTTP requests from users (actually from the user's clients) to place call back calls and also to perform management tasks concerning both users and the IP-PBX.

4.4.1 Using EJB 3 replacing EJB 2

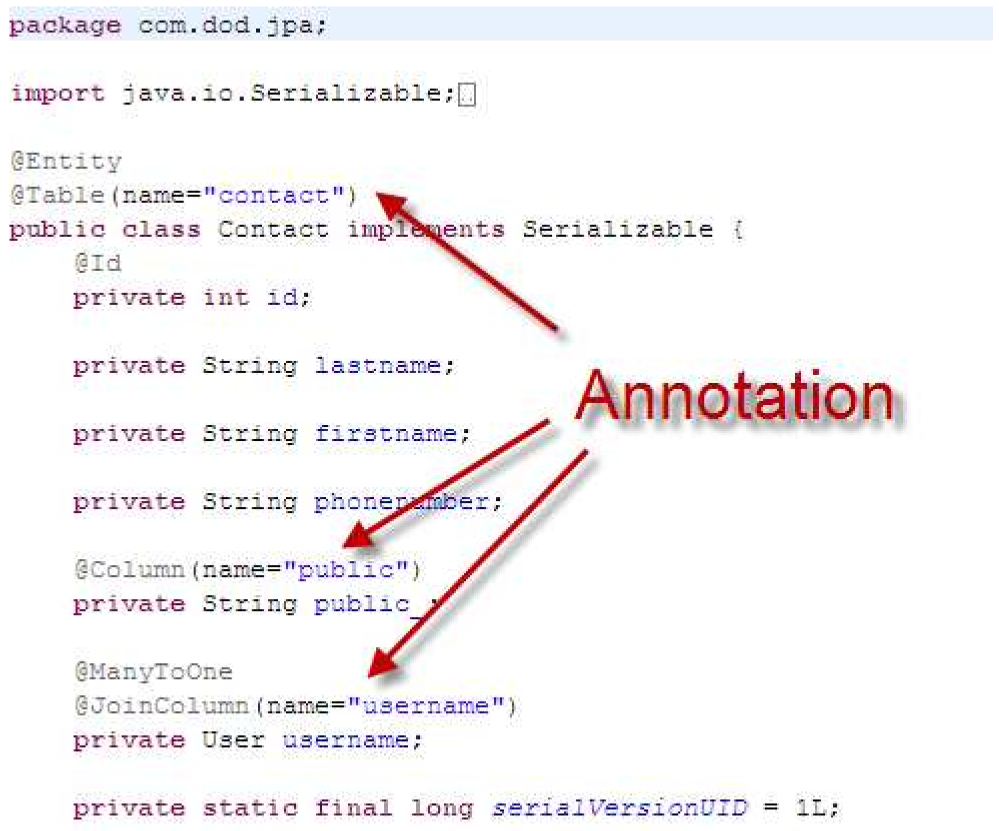
As mentioned at subsection 2.3.1, EJB is a type of server side component. All the logic and operations on the database are handled by EJBs. The existing DoD server used EJB version 2. In 2006, EJB version 3 was released. Compared to EJB 2, this version is easier to write and it also offers several improvements. Those improvements that are of specific interest for DoD are described in the following paragraphs.

4.4.1.1 Simplify writing EJB

In EJB verison 2, we need two interfaces and one implement class for an EJB. Although some integrated development environments (such as JBuilder and Eclipse) can generate these interfaces and class automatically, it is still a little bit complex for developers when developers need to write many EJBs. In version 3, the developer only needs to write one interface and a class. This is similar to the usual Java programming, thus the developers can write and manipulate these EJBs like normal classes.

4.4.1.2 Using annotation instead of XML descriptor

In EJB version 2, each EJB needs a behavior definition in a XML descriptor (ejx-jar.xml). If you want to change the behavior of an EJB, you may have to change both the code and the XML descriptor. In version 3, you do not need to have a XML descriptor for each EJB, instead you can use annotation to define the behaviors of EJBs. Annotations provide information about a program, but are not part of the program itself. They have no direct effect on the operation of the code as they are simply annotations.[34] So by using annotation, you can annotate your code rather than writing XML descriptors in another file, see figure 22.

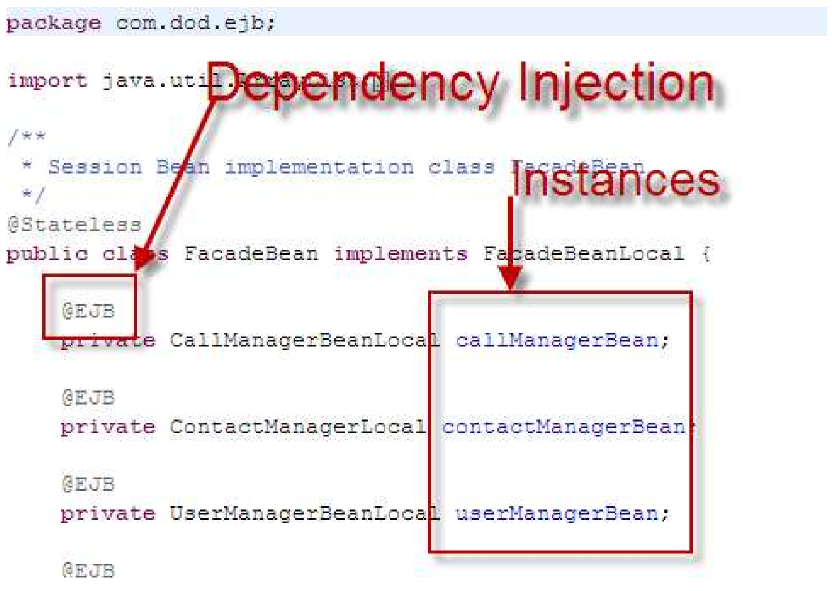


```
package com.dod.jpa;

import java.io.Serializable;

@Entity
@Table(name="contact")
public class Contact implements Serializable {
    @Id
    private int id;

    private String lastname;

    private String firstname;

    private String phonenumber;

    @Column(name="public")
    private String public_;

    @ManyToOne
    @JoinColumn(name="username")
    private User username;

    private static final long serialVersionUID = 1L;
```

Figure 22. Annotation for EJB3

4.4.1.3 Using Dependency Injection instead of JNDI look up

When using EJB 2, if you want to have an instance of an EJB, you have to use JNDI look up to get an instance. This is a very repetitive operation for the developer. In EJB 3, you can use Dependency Injection to get an instance. You simply write an annotation above the class members. When the class is initialized, the container will automatically get an instance for this class member. See figure 23



```
package com.dod.ejb;

import java.util.

/**
 * Session Bean implementation class FacadeBean
 */
@Stateless
public class FacadeBean implements FacadeBeanLocal {

    @EJB
    private CallManagerBeanLocal callManagerBean;

    @EJB
    private ContactManagerLocal contactManagerBean;

    @EJB
    private UserManagerBeanLocal userManagerBean;

    @EJB
```

Figure 23. Dependency Injection for EJB3

4.4.2 Using model-view-controller mode 2

The structure of the existing DoD server was roughly two layers. The first layer is a presentation layer which is used for showing GUIs, the second layer was an EJB layer used for all logic and database manipulations. This is basically model-view-controller model 1 architecture. Model 1 is the most common way in the web application development from the emergence of the web. Entry point for every request starts from JSP. Model 2 is the new approch to overcome the shortcomings of Model 1. MVC(Model View Controller) which is the fundmental of Model 2, is the concept which has been used in the GUI development such as Applet. There are already many web framework with MVC in place these days. JSP is the entry point in Model 1, whereas Controller(Servlet) is the entry point in Model 2[35]. Also some pages are not well organized, because some logics which should be in the business layer are also in the presentation layer. For example, one JSP page includes almost 1000 lines of code.

In order to make the code easy be to maintained, we changed the structure to model-view-controller model 2 architecture. This model has three layers, with each layer having its own responsibility. The difference between a model 1 and model 2 architecture is that the request from users is not sent directly to the presentation layer. Instead there is a controller that receives request from the user and pass this request to the business layer or directly to presentation layer. By using model-view-controller model 2 architecture, we can reduce the amount of logic code in the presentation layer, because the controller can process the requests and get information needed for the presentation layer. Therefore, a model-view-controller model 2 architecture is better than a model-view-controller model 1 architecture, especially for web applications.

4.3 Using one GUI for both calling and configuring

The existing DoD solution is controlled by two different web applications. One is used for controlling the DoD server and the other is FreePBX[36] for configuring the IP-PBX (Asterisk). This is inconvenient for the administrator. For example, if the administrator wants to add a user to the DoD sever, then the administrator has to create an account in the DoD server for receiving requests from this user and create a SIP account in the IP-PBX by using FreePBX, then the administrator needs to specify the inbound route for this user. Even worse, if the administrator needs to enable or disable the followme feature for this user and they could not manually change this configuration using FreePBX. However, the user might want their followme setting changed frequently.

Another problem with using separate web applications to control the DoD solution and the IP-PBX is that FreePBX has its own format for configuration files. If we want some feature but FreePBX has not implemented this feature, we cannot manually configure those configuration files. If we change any settings of Asterisk by using FreePBX, then we have to reload the whole Asterisk configuration to make these settings take effect. We can save all the settings in database, but FreePBX does not support saving settings in a database.

For these reasons, we created a single GUI for both placing calls and configuring Asterisk. For example, if an administrator wants to add a new user, he or she only needs to create an account in the DoD server. The DoD server will automatically change the appropriate settings of Asterisk.

Chapter 5 Evaluation

This new DoD solution will be released as a commercial product to replace the existing DoD solution. The functionality and performance of the system are very important for users. In order to quantify the system's performance and to identify weak points that can be improved in the future, we tested several aspects of this version of our solution. The following sections will explain in detail some of the tests that were conducted.

5.1 Test Environment

The test environment includes the test machines, their operating systems, the software, network condition, and hardware SIP phones. Specifically the test environment consisted of:

- Test machines
  - Desktop computer, CPU: Inter Celeron 2.53GHz, Memory: 1GB, Hard Driver: 400GB SATA, Operating System: Fedora 9 Kernel Linux 2.6.27
  - Desktop computer, CPU: Inter Pentium 2.8GHz, Memory: 1GB, Hard Driver: 80GB SATA, Operating System: Windows XP Professional Service Pack 3
- Network configuration

  One of the test machines which was the DoD server. This computer has a public IP address. Another test machine is behind a NAT on a private IP network. The router of this second machine is connected to the same Internet Service Provider (ISP) as used by the first test machine. See figure 24.
- Software
  - Web Container and Application Server: JBoss Application Server 5.0.1-GA
  - IP-PBX: Asterisk 1.4.24.1
  - Java Runtime Environment: 1.6.0_14
  - Testing tools: LoadRunner 8.1 Feature Patch 4, Jakarta JMeter 2.3.4
- Hardware SIP phones
  - Hitachi-Cable Wireless IPC-5000AE WiFi SIP Phone

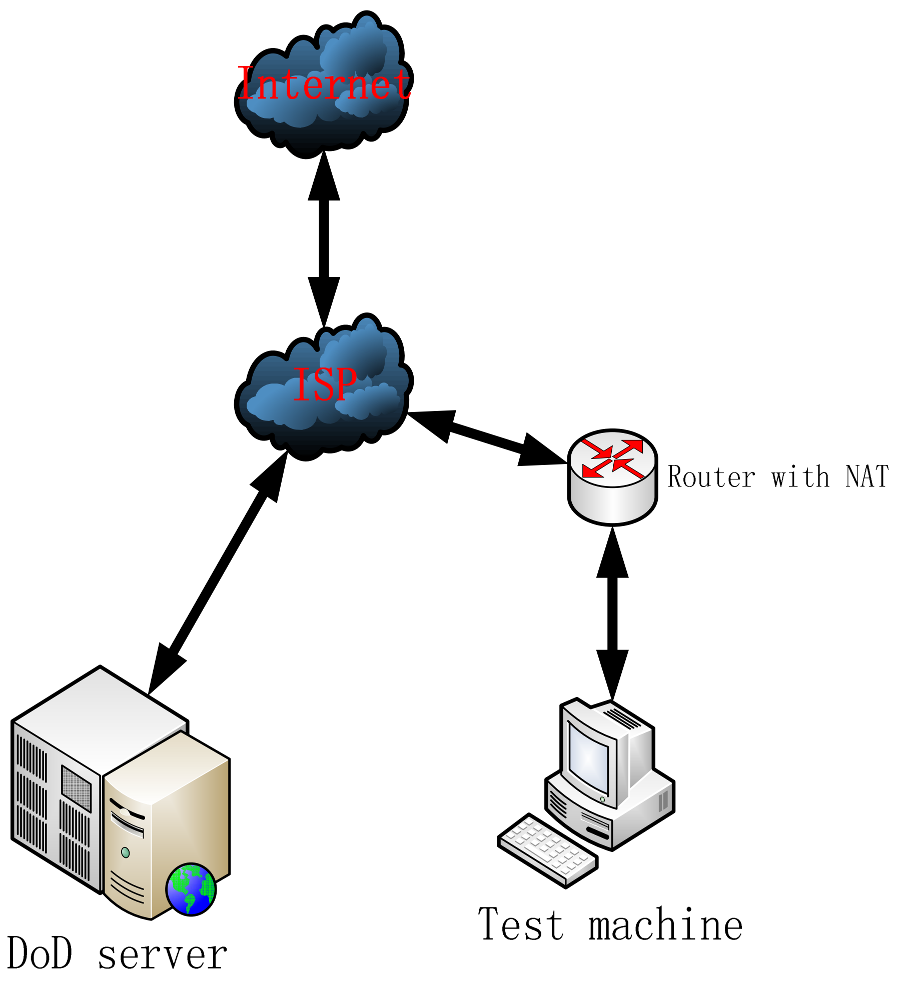


Figure 24. Network topology of test environment

5.2 Tests for JBoss Application Server (DoD server)

The DoD server is a web application running in a JBoss application server. This DoD server is used for processing request from users, showing GUIs and responding to user interface commands, and sending requests to Asterisk to place calls. A key performance metric for the DoD server such is how many users it can handle simultaneously or how many requests can process per second. This performance metric is very important for both users and developers. Additionally, understanding the performance of this new implementation will help us to analyze where we could improve the system in the future.

5.2.1 Test plan

In order to make this test as realistic as possible, we record the actions of a user in a script. This user script will do normal actions as a form of “smoke testing”. Smoke testing is a "shallow and wide" approach to test an application. The test script "touches" all areas of the application without getting too deep.[37] Because this smoke testing tests all the main functions of the system, we get only a general impression of the system's functionality. After the first script has successfully be run, then we will use a testing tool called “LoadRunner”[38] to simulate several users in order to test the overall load that the DoD server can support. Following this testing, we analyze the results.

As mentioned in section 5.1, the test machine, where we put the DoD server, has a public IP address. We installed LoadRunner on another physical machine. There is an Internet connection between these two machines. The LoadRunner emulates multiple virtual users. Each of these virtual users will execute a script we previously recorded. As a result of emulating multiple users, many requests will be sent to the DoD server, and the LoadRunner will record the results for our later analysis.

5.2.2 Test tool

HP’s LoadRunner is set of applications that provide businesses a way to reliably simulate the load of multiple users against their given application. LoadRunner emulates hundreds of thousands of concurrent users to apply production workloads to almost any client platform or environment. LoadRunner stresses an application from end to end applying consistent, measurable, and repeatable loads. It then uses the data to identify scalability issues that can affect real users in production.[39]

The LoadRunner can be used for testing different kinds of applications on different platforms. In our case, we use is to test a web application. The LoadRunner tests the web application by recording and replaying the HTTP or HTTPS requests sent from each virtual user and checking the responses from the web server. There are four steps to testing using LoadRunner.

1. Record the actions of a representative user into a script
2. Replay and verify the script to make sure it can be run correctly
3. Create a scenario with a specified number of users, defining the iteration, time schedule, and so on for this scenario
4. Running this scenario as a test

We assume that 25% users of all the users are always calling via the DoD solution. Because the SIP trunk that we are using can support 50 simultaneous calls, this means that the largest number of users that our tests environment can handle is 200 users.

5.2.3 Test results

The first part of this test is to make sure the DoD server is stable while supporting a certain number of users (in this test: 200 users). So we conducted a test that simulates 200 users. From the user's point of view, the system's performance is how long the user needs to wait for each request before getting a response. A typical rule of thumb is that if the response time for pages is within 2 second then the user perception is that the system is perfect, between 2 and 5 seconds is normal, and having to wait more than 5 seconds is bad. In the following plots, Vusers is the number of virtual users. Figure 25 shows the number of users in the system as a function of time. As can be seen the number of users increases by 10 users per second after an initial start up time (roughly 5 seconds). Each user's call last for approximately 10 seconds. As can be seen in this figure, after making a call the user leaves the system. After 115 seconds all users have made calls and have left the system. Note that the values shown as diamonds in the plot are from the data which was collected every tenth of a second. This will also be the same period over which results are averaged in the next subsection.

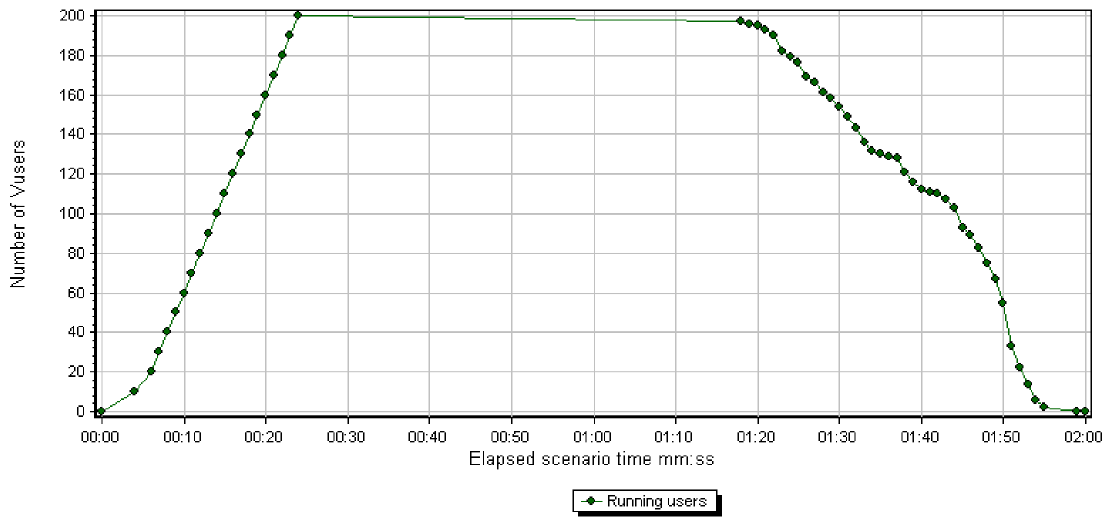
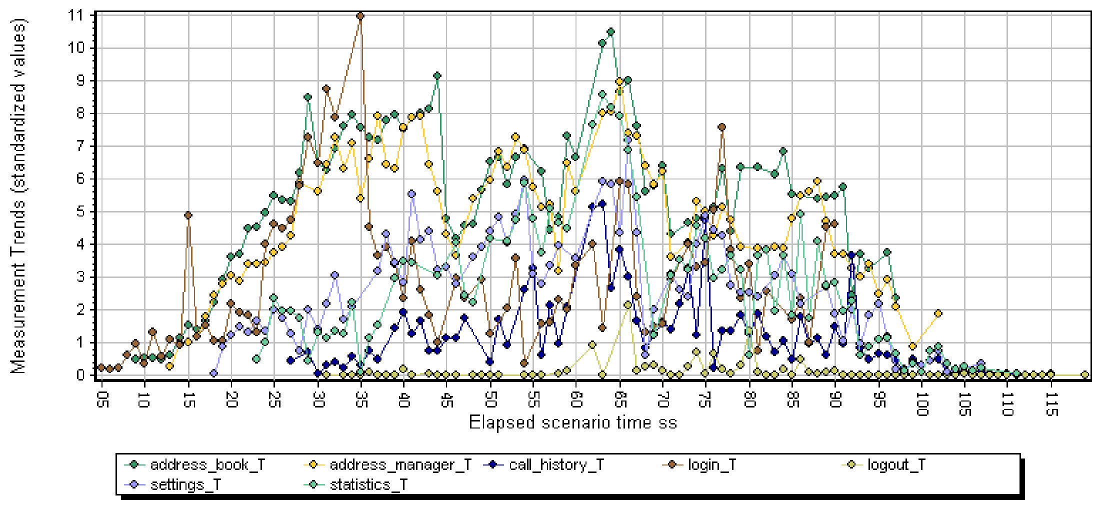


Figure 25. Running users chat

5.2.4 Analysis of the test's result's

In this test, no errors occurred. This means the DoD server was stable while handling 200 users with the 25% active ratio and the average call duration of 10 seconds. In figure 26, each curve represented the average response time (in seconds) for each individual page utilized by a user when using the DoD sever. The averages are computed every tenth of a second. The minimum value of the average response time of each page is 0.12s this is the response time when the user is logging out. The maximum value of the average response time is 5.38s when users request their address book page. The average response time for all pages is 2.84s. This test indicated that this configuration of the DoD server can serve about the 200 users, but the response time of each page is almost three seconds - so the users perception will be that the system is a bit sluggish - although it is still "normal". However, the user must wait for a long time, which is not be accepted. For example, after the test has been running for roughly 35s (thus all 200 users are in the system), the response time to get the user's address book page is on average 10.44s; this will be perceived as very bad performance. Hence the system cannot successfully support 200 simultaneous users - with the utilization model that we have used for this testing. These test results suggest that the system could support roughly 150-160 simultaneous users with less than 5 second response times. Unfortunately, this means that we will only be making ~75% utilization of our SIP trunk. Hence the server's performance needs to be scaled up so that we could server all 200 users while providing less than 5 second response times.



Color	Measurement	Minimum	Average	Maximum	Median	Std. Deviation
🟩	address_book_T	0.493	5.388	10.444	5.496	2.29
🟨	address_manager_T	0.264	4.838	8.941	5.123	1.964
🟦	call_history_T	0.036	1.218	5.203	0.932	1.163
🟫	login_T	0.178	2.959	10.937	2.367	2.16
🟨	logout_T	0.007	0.128	2.153	0.012	0.32
🟪	settings_T	0.061	2.675	7.177	2.596	1.593
🟩	statistics_T	0.053	2.687	8.549	2.343	2.06

Figure 26. Average response time for each page (in second)

5.2.5 Problems faced in Tests

During the testing we faced some problems. When there are more than 200 users, the DoD server throws many “java.net.SocketException: Too many open files” exceptions. This is because Linux has a limit for the maximum number of open files. By default, Linux allows 1024 open files at the same times. We can use command “ulimit –n” to check this limit and use “ulimit –n [num]” to set it. We should examine what would be a better setting for this limit. There is also a limitation in JBoss, as the configuration file “server.xml” defines how many connections JBoss can handle. The default value is 200, thus we should increase this value to enable testing with more than 200 users. As noted in the previous section the response times with more than 150-160 users are unacceptable, thus neither of these parameters limited our testing. However, for future tests with a more powerful server these limits must both be increased.

5.3 Test for MySQL

The database has an important role in DoD solution, as it is used by both the DoD server and Asterisk. An advantage of this approach is that users can change their own settings by simply changing their entry in the database. Additionally, Asterisk validates the users by checking the database. Therefore out next test is designed to test the performance of database.

5.3.1 Test plan

Because LoadRunner cannot test MySQL, we use another test tool: Apache's "JMeter" (details of this tool are given in the next subsection). As mentioned earlier, this DoD server was expected to handle roughly 200 users. In the previous test we measured the performance when accessing each of the 7 different pages that a user would normally use. We assume that each page requires 2 database queries and each page will be required twice. Because that in almost all pages, the information showed to users may be not in the same table of database, thus we normally need to query database to get all data we need. Therefore, in this test we measure the effects when 200 users each have to query the database 4 times.

5.3.2 Test tool

Apache's JMeter is open source software. It is a 100% pure Java desktop application designed to load test functional behavior and measure performance. It was originally designed for testing web applications, but has since expanded to other test functions.[40] Just like HP's LoadRunner, we can define users which are called a thread group in JMeter. Next we specify one or more JDBC requests to perform for each of this thread groups. The final element of the test plan is a Listener. The listener is responsible for these all of the results of these JDBC requests in a file and presenting a visual model of the data.[41]

5.3.3 Test result

Figure 27 shows the results of each simulated user making 4 queries with an average of xx seconds between query requests.

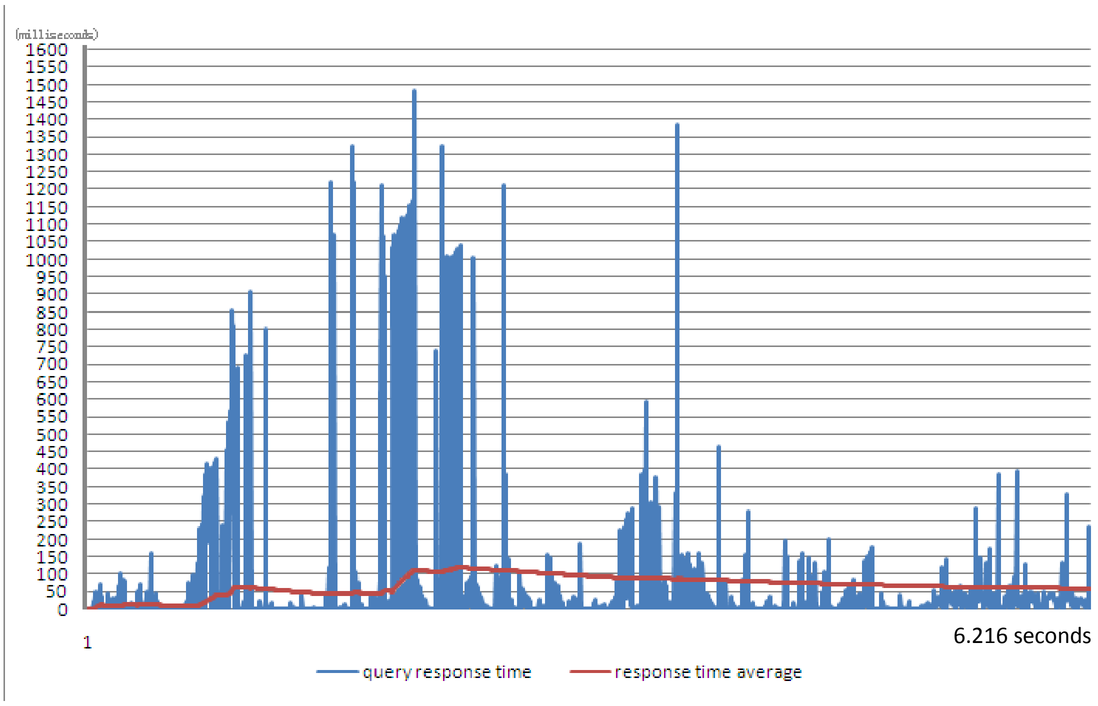


Figure 27. Response time to query the database

This testing includes 2800 queries (based upon 200 user × 7 pages × 2 queries) of database. The average time of database query takes 58.961ms. The test duration was 6.216 seconds, this was enough time to complete all the queries for all 200 users. Note that despite the queries being spread over more than six seconds, the maximum query response time is more than 1.450 seconds, which leaves very little other time for the system to perform other operations while maintaining a response time as seen by the user of less than 2 seconds.

5.3.4 Test Analyze

There were no errors reported during this test. All the queries got their correct responses. While the average time to perform a database query was 58.961 milliseconds, which is shorter than the response time of pages, the maximum query times are not so encouraging. Therefore further testing is needed to understand who to reduce this maximum query delay by an order of magnitude (or more). Since the DoD server, Asterisk, and the database are running in the same machine, the database queries are not actually send out on the network as Ethernet frames. Hence we have not measured the performance that would be experienced if these servers were actually running on different computers. Adding this additional delay to each of these queries would increase the delays by one or more milliseconds, even with a 100 Mbps ethernet between the machines. Comparing this with the 2.2 milliseconds average time per query (for 2800 queries in 6.216 seconds) we see that an increase of one or two milliseconds for the network processing could double this average time! Additionally, comparing this average time over the whole period with the average time delay for the actual queries, we see that the distribution of query times has both a long tail and is more than 20 times the simple average over the whole period.

5.4 Evaluation of Asterisk and the SIP trunk

Asterisk also queries the database, when checking the caller ID or finding out which caller ID should be presented. This also contributes to the load on the database. The SIP trunk that we have used for testing supports 50 calls at the same time. Thus Asterisk could potentially have to check the call IDs of 50 simultaneous calls and it must also retrieve the correct caller IDs from database to present to users for all of these calls. Each call will generate 2 database queries. As a result, there are 100 queries that could be sent at the same time. Comparing the number of requests sent to the 2800 requests sent by the DoD server, one might think that these additional database requests would not be a problem. However, this is not true; since the load testing of queries to the database by the DoD server were spread over more than six seconds, while the maximum load of the SIP trunk could all come within less than one second! Therefore if the calls come in during a short period of time, there may be a performance limit caused by the SIP trunk.

There is also a problem with the assumptions about the SIP trunk and our performance measurements. Each call coming into the DoD server could cause one or more calls to go out. For example, if the callee has a fixed phone and a cellular phone and is using the FollowMe functionality, then two outgoing calls have to be made. Depending upon whether we are using early media or not - this may require 0, 1, or 2 calls capacity from the SIP trunk.

If the caller is coming in via the PSTN (or PLMN) and placing a call that must also go out via the PSTN(or PLMN), and if these calls are carried via the SIP trunk, then because of call bridging we need to allocate SIP trunk resources for the duration of these two calls.

We have not properly tested the performance of the Asterisk server or SIP trunk, hence these should be tested more thoughoutly in future tests. Additionally, we might be able to leverage the testing that others have done of Asterisk performance when connected to a SIP trunk, such as [42] and [43].

5.5 Conclusions

We have tested the DoD server and the database, by see that although they are stable enough to support for 200 users, the performance of the system is not as good as we might hope. However, the system is capable of supporting 150-160 users (with our scenario parameters). Additionally, we see that the database performance while sufficient on average, is not acceptable for all users. Thus we need to examine how this could be improved. Further more the performance of the DoD sever is poor, as the response time for each page is too long in some cases. In the future the performance should be improved by: optimizing the code, changing the web container (such as WebLogic[44] and WebSphere[45]), and using a more powerful server. The least expensive method of improving the performance of the system to meet the needs of 200 users would be to use a more powerful server, as all of the current code could be run. Alternatives, such as partitioning the services over multiple servers might actually decrease the overall system performance.

Chapter 6 Conclusions and Future Work

The project accomplished the goal for this thesis project, but new needs for improvements were found. In the first section of this chapter we will summarize our conclusions and in the second section as summary of potential future work is given.

6.1 Conclusions

The goal of this thesis project was to improve the server side of the prior DoD solution. The complete DoD solution provides two main methods to place calls via mobile devices. Based upon earlier user tests, there were some problems with the existing DoD server. Additionally, we found some other aspects of the existing solution that could be improved. In order to provide a more stable and user-friendly solution for users, we made some changes and improvements in both the IP-PBX and DoD server. In particular we improved the dialplan. By using ARA, the dialplan and SIP settings are saved in a database. This makes it possible for applications to control and manage Asterisk. Additionally, changes in settings of Asterisk can be made without reloading. Another improvement was optimizing the caller ID by using a local channel and the command "NoCDR", this enables Asterisk to deliver the correct caller ID to different kinds of devices at the same time. At the same time, our new configuration of Asterisk can filter call records, keeping only the relevant CDR information in the database. We have also improved the graphical user interface to show call data records to users and administrators. The final major improvement was to rewrite the DoD server using EJB3, making it both easier to write EJBs and reducing the number of lines of code that need to be written and maintained.

The new improved server is working correctly with the Symbian client (this client is the topic of another master's thesis). However, there are some aspects of the system that could be improved in the future. These changes are described in the following section.

6.2 Future work

Today the new DoD server is up and running, but it is not prefect and still has some problems. For example, a caller can cheat the IP-PBX by faking its caller ID, the caller may have to wait a long time to place a call, and so on. Although there are many areas for future work to improve this new DoD solution, in the following subsections we will describe some of the most urgent areas for improvement.

6.2.1 Using a framework in the presentation layer

The presentation layer of the DoD server is composed of many JSP pages. However, maintaining a web service is only feasible when the web application does not contain many pages. However, in the future the DoD server may include even more logic and pages, thus it would be better to use a framework to manage and control the complete presentation layer.

Presentation frameworks such as Struts[46] and JSF[47] are well designed for implementing the presentation layer. These are already useing a model-view-controller model 2 architecture, and these frameworks can help the developer reduce the amount of logic needed in the presentation layer and provide internationalization support. This internationalization support is particularly important for the product as one of the major uses of the DoD solution is with DoD servers located in multiple countries in order to reduce the cost of roaming.

6.2.2 Multiple access numbers for the IP-PBX

The current IP-PBX used in the DoD solution has only a single access number for call through, no matter what kind of client is using this number. When placing call through calls, the client needs to send DTMF tones, but the speed of sending DTMF is different in different kinds of clients. For example, in the Symbian client, the speed of sending the necessary information via DTMF can be very fast. However, when sending DTMF tones using a client written in JavaME, the client send a request to OS of the mobile phone, then the mobile phone itself sends these DTMF tones for client. As a result the client cannot control the speed of sending DTMF tones, so different clients take different amount of time to send the required information. However, when the IP-PBX is receiving a call through request the IP-PBX uses a timeout to decide whether the client has finished sending the required information. If the IP-PBX only has a single access number, then the timeout must be set to be the time required for the slowest client, but a user with a faster client will wait unnecessarily in order to place call through calls.

One solution is to use multiple access number for the IP-PBX. For example, a very short timeout can be used for the access number used by Symbian clients, a long timeout at another access number for Java clients or other clients. Finally, a very long timeout can be used for users manually entering this information via the keypad of their phone. Alternatively, a new JavaME client could be written that directly generated the combinations of the two tones needed for the DTMF dialing, thus allowing rapid signaling of the call through number for both clients.

6.2.3 Other ways to place call back calls (SMS and Voice)

Currently we send an HTTP request to the DoD server to tell it the caller and callee's numbers, and to provide a username and password. However, in some countries it is expensive to send information via a mobile phone's Internet connection. Fortunately, there are several alternative for placing call back calls. Two of the most obvious are: SMS and voice. These two alternatives are described below.

6.2.3.1 SMS call back

Another means of providing the IP-PBX with the caller and callee's number, password, and username would be via a SMS. The IP-PBX or DoD server could receive these SMSs messages, Asterisk can connect to a GSM gateway or a mobile phone (or bank of mobile phones) could be used to receive SMSs. Similarly the SMSs could be received by the DoD server. There are many SMS "providers" that provide a service for receiving SMSs via a web interface. As a result the DoD server could check for incoming SMSs by sending a request to such a provider or the provider can send a HTTP (POST) request including the content of the SMS to us. By using these two methods, the DoD server or IP-PBX can get the information necessary to place the call back calls.

6.2.3.2 Voice call back

There is another way to place a call back call. The caller could place a call to the IP-PBX, the IP-PBX would hang up the call. Because of the incoming call attempt, the IP-PBX knows the number of the caller from the incoming caller ID. Given this information the IP-PBX can place a call back to the caller. After the caller answers this call, the caller tells the IP-PBX the number of the callee by sending DTMF tones or by using voice commands to indicate who is to be called. Once the IP-PBX knows the necessary information, the IP-PBX places another call to callee. After the callee answers this call, the IP-PBX bridges these two calls as usual for the call back service.

6.2.4 Increasing security for call through

Today the IP-PBX can be cheated by faking a valid caller ID. In order to make call through more secure, we could add a password (using only digits) for each user. When the IP-PBX answers the call through call, the caller could be prompted to enter their password. This approach would make call through more security, but will increase the delay in initiating a call. This will also require a change in the logic for call through for the Symbian (and other) clients.

6.2.5 Providing machine readable call history

In many cases it is not a human that will process the call history, but rather a program that is going to transform this call history into some other form or mine this call history for information. As an example of the first, the user might want to get their call history and automatically generate a list of contacts that they have not called within the last month - in order to keep up to date with their customers. As an example of the later, a user might want to have an application that automatically generates entries in the user's calendar to call those persons who called them when they were unavailable.

References

[1] Max Weltz, Dial over Date solution, Master thesis, Royal Institute of Technology (KTH), School of Information and Communication Technology, Stockholm, COS/CCS 2008-2, Sweden, February 2008, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080221-MaxWeltz_ExjobbReport-with-cover.pdf

[2] VOIP wikipedia,[www] http://en.wikipedia.org/wiki/VoIP Last access on 2009-5-25

[3] Skype, [www] http://www.skype.com, Last access on 2009-08-02

[4] Philippe Biondi and Fabrice Desclaux,"Silver Needle in the Skype", BlackHat Europe, March 2nd and 3rd, 2006 http://www.secdev.org/conf/skype_BHEU06.handout.pdf

[5] RTP wikipedia, [www] http://en.wikipedia.org/wiki/Real-time_Transport_Protocol Last access on 2009-5-25

[6] SIP wikipedia, [www] http://en.wikipedia.org/wiki/Session_Initiation_Protocol Last access on 2009-5-25

[7] MiniSIP, [www] www.minisip.org Last access on 2009-5-25

[8] X-lite, [www] http://www.counterpath.net Last access on 2009-5-25

[9] SJphone, [www] http://www.sjlabs.com Last access on 2009-5-25

[10] Nonoh, [www] http://www.nonoh.net Last access on 2009-08-02

[11] VBuzzer, [www] http://www.vbuzzer.com Last access on 2009-08-02

[12] VoIP Stunt, [www] http://www.voipstunt.com Last access on 2009-08-02

[13] Fring, [www] http://www.fring.com Last access on 2009-08-02

[14] Asterisk, [www] http://www.asterisk.org Last access on 2009-08-02

[15] Asterisk wikipedia, [www] http://en.wikipedia.org/wiki/Asterisk_(PBX) Last access on 2009-5-25

[16] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler SIP: Session Initiation Protocol, IETF, RFC 3261, June 2002.

[17] Henry Sinnreich and Alan B. Johnston, Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol, 2nd Edition, Wiley, August 2006, ISBN: 0-471-77657-2

[18] Luan Dang, Cullen Jennings and David Kelly, July, 2002, Practical VoIP using VOCAL, O'Reilly, ISBN 10: 0-596-00078-2 | ISBN 13: 9780596000783

[19] voip-info, [www] http://www.voip-info.org/wiki/view/Asterisk+sip+type Last access on 2009-5-25

[20] Jim Van Meggelen, Leif Madsen, Jared Smith, September 2005, Asterisk: The Future of Telephony, O'Reily, ISBN 10: 0-596-00962-3 | ISBN 13: 9780596009625

[21] voip-info, Asterisk CDR Fields, [www] http://www.voip-info.org/wiki/view/Asterisk+billing Last access on 2009-06-18

[22] Sun, [www] http://java.sun.com/javaee/ Last access on 2009-5-25

[23] Kevin Mukhar and Chris Zelenak with James L. Weaver and Jim Crume, Beginning Java EE 5: From Novice to Professional, Apress, October 2005, ISBN10: 1-59059-470-3 | ISBN13: 978-1-59059-470-4

[24] Wu Xiao, SIP on an Overlay Network, Master thesis, Royal Institute of Technology (KTH),

School of Information and Communication Technology, Stockholm, Sweden, (expected) August 2009

[25] Sun, [www] http://java.sun.com/products/ejb/ Last access on 2009-5-25

[26] Raghu R. Kodali and Jonathan Wetherbee with Peter Zadrozny, Beginning EJB™ 3 Application Development: From Novice to Professional, Apress, September 2006, ISBN10: 1-59059-671-4 | ISBN13: 978-1-59059-671-5

[27] Debu Panda and Reza Rahman and Derek Lane, EJB 3 in Action, Manning Publications Co., April 2007, ISBN: 1-933988-34-7

[28] JBoss Application Server Official Documentation Page, JBoss Application Server 5, Installation and Getting Started Guide, [www] http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Installation_And_Getting_Started_Guide/5/html/Book-Preface.html Last access on 2009-5-25

[29] Companhia de Telecomunicações de Macau S.A.R.L. (CTM), [www], http://www.ctm.net Last access on 2009-08-02

[30] CDR wiki, [www] http://www.voip-info.org/wiki/view/Asterisk+cdr+csv Last access on 2009-08-02

[31] Asterisk config extensions.conf, [www] http://www.voip-info.org/tiki-index.php?page=Asterisk%20config%20extensions.conf Last access on 2009-08-02

[32] Asterisk local channels, [www] http://www.voip-info.org/wiki/view/Asterisk+local+channels Last access on 2009-08-02

[33] NoCDR, [www] http://www.voip-info.org/wiki/view/Asterisk+cmd+NoCDR Last access on 2009-08-02

[34] Annotations, Sun, [www] http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html Last access on 2009-08-02

[35] Java Basics, Model 2, [www] http://javias.net/blog_mu/blog/2009/04/16/java-basics-model-1-and-model-2/ Last access on 2009-08-17

[36] FreePBX, [www] http://www.freepbx.org Last access on 2009-08-02

[37] Smoke testing, [www] http://en.wikipedia.org/wiki/Smoke_testing Last access on 2009-08-18

[38] Load Runner, HP, [www] https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17%5E8_4000_100__ Last access on 2009-08-18

[39] LoadRunner to Performance Center Migration, An Objective Analysis of Migration Options White paper, HP, 4AA1-4912ENW, May 2008

[40] JMeter, Apache, [www] http://jakarta.apache.org/jmeter/index.html Last access on 2009-08-19

[41] User Manual, JMeter, Apache, [www] http://jakarta.apache.org/jmeter/usermanual/build-db-test-plan.html Last access on 2009-08-19

[42] Asterisk V1.4.11 Performance, TransNexus, [www] http://www.transnexus.com/White%20Papers/asterisk_V1-4-11_performance.htm

Last access on 2009-08-28
[43] Asterisk Background performance test, KOLMI soft, [www] http://wiki.kolmisoft.com/index.php/Asterisk_Background_performance_test Last access on 2009-08-28
[44] Oracle, WebLogic, [www] http://www.oracle.com/appserver/index.html Last access on 2009-08-20
[45] IBM, WebSphere, [www] http://www-01.ibm.com/software/websphere/ Last access on 2009-08-20
[46] Apache Struts, [www] http://struts.apache.org/ Last access on 2009-08-02
[47] JavaServer Faces Technology, [www] http://java.sun.com/javaee/javaserverfaces/
Last access on 2009-08-02

TRITA-ICT-EX-2009:104