Developing a Mobile Extension Application

OptiCaller Application and Provisioning System

TAO SUN

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:177

Developing a Mobile Extension Application

OptiCaller Application and Provisioning System

Masters thesis

Tao Sun

<taos@kth.se>

2009-10-15

Examiner:

Professor Gerald Q. Maguire Jr.

Industrial Supervisor:

Jorgen Steijer, OptiCall Solution AB

School of Information and Communication Technology
Royal Institute of Technology

Abstract

Today companies (especially large companies whose employees make a lot of international business trips) often have very large telephone bills. While international roaming technically works with GSM, the cost of phone calls from one country to another are often much higher than calls within a country. Despite political pressure to reduce the costs of roaming within the European Union, the cost of phone calls from one country to another are often quite expensive. A cost-saving solution is eagerly desired by many firms. OptiCall Solutions AB has designed an integrated system called the Dial over Data (DoD) solution. In this scheme, a caller sends a data request to the DoD server instead of directly making a call to the party that they wish to speak with. The DoD solution uses an Internet Protocol Private Branch Exchange (IP-PBX) to make two Session Initiation Protocol (SIP) calls, one to the caller and another to the callee, then bridges these two calls. To realize cost savings, the cost of these two calls and the cost of the infrastructure necessary to make the two calls and to bridge them must be lower than the cost of the direct call.

Call Through is a service for making cheap international calls which is provided by many telecommunications companies. Instead of making a direct call to the party the caller wants to speak with, the caller makes a call to the local call-through service access number. The caller dials the actual callee's number after the call is established. The service provider then makes a call to the callee and bridges the two calls. The caller only needs to pay for a local call and the service subscription fee, rather than the expensive international call fee. This can greatly reduce the costs for user who need to make a lot international calls. Today, many companies use such a call-through service in order to reduce the total cost of their employees' calls.

Additionally, the Mobile Extension (MEX) is a concept which gives mobile users the ability to use their mobile phone in the same manner as their fixed office phone, for example, by providing services such as setting presence and transferring a call. In addition, the user should experience a consistent interface, for example the calling number displayed for the callee should always be a number that the callee could use to return the call.

In this thesis project, a mobile extension application called 'OptiCaller' based on the Symbian OS was developed and evaluated. This application is part of the DoD solution and provides client side functionality. Furthermore, it supports making call-through calls and using the MEX functions. Additionally, it is designed and implemented in a flexible way so that it can work with a variety of different PBX solutions.

A provisioning system called 'OptiCaller Provisioning System' was also designed and evaluated. This provisioning system was tailored for the 'OptiCaller'. It provides administrators a platform to manage the OptiCaller application on the end-users' mobile phones.

Sammanfattning

Idag har företag (i synnerhet stora företag vars anställda gör en hel del internationella affärsresor) ofta mycket stora telefonräkningar. Även om internationell roaming fungerar tekniskt i GSM så är kostnaden för telefonsamtal från ett land till ett annat ofta mycket högre än samtal inom ett land. Trots politiska påtryckningar för att minska roamingkostnaderna inom EU, så är kostnader för telefonsamtal från ett land till ett annat ofta ganska dyra. En lösning som kan minska dessa kostnader välkomnas av många företag. OptiCall Solutions AB har utvecklat ett integrerat system som kallas Dial över Data (DoD). I detta system skickas först en samtalsbegäran från en klient till en server istället för att klienten direkt ringer ett samtal till den som de vill tala med. DoD systemet använder Internet Protocol Private Branch Exchange (IP-PBX) för att göra två Session Initiation Protocol (SIP) samtal, en till den som ringer och en den uppringda parten. Sedan kopplas dessa två samtal ihop. Värt att beakta är att kostnaden för denna infrastruktur samt att producera två samtal i stället för ett bör vara lägre än kostnaden för ett direkt samtal.

Samtalskort är en tjänst för att göra billiga internationella samtal vilket erbjuds av flera telekommunikationsföretag. Istället för att göra ett direkt samtal till den som den uppringande vill tala med, gör först ett samtal till den lokala call-service noden. Med hjälp av en transparent klient så behöver användaren inte själv ringa call-service noden utan det sker helt automatik. Väl besvarat av call-service noden så kopplas samtalet till den destination användaren vill ringa till. Den som ringer behöver bara betala för ett lokalsamtal och tjänstens prenumerationsavgift. Detta kan kraftigt minska kostnaderna för användare som gör en hel del utlandssamtal. Idag är det även många företag som tillhandahåller en sådan funktion till sina anställda för att minska den totala kostnaden för deras samtal.

Mobile Extension (MEX) är ett begrepp som ger mobila användare möjlighet att använda sin mobiltelefon på samma sätt som sina fasta kontorstelefon, till exempel genom att tillhandahålla tjänster såsom inställningen närvarostatus och koppling ett samtal. Användaren bör uppleva ett konsekvent gränssnitt, till exempel att det uppringande numret som visas alltid bör kunna användas för att ringa tillbaka till den som ringt.

I detta examensarbete har en så kallad mobil anknytning tillämpning "OptiCaller", baserad på Symbian OS, utvecklats och utvärderats. Denna applikation är en del av DoD lösningen och ger funktionalitet på klientsidan. Den har även stöd för samtalskortfunktioner och MEX funktioner. Den är dessutom utformad på ett flexibelt sätt så att den kan arbeta med en rad olika PBX lösningar.

Ett system för provisionering anpassat för OptiCaller kallat ”OptiCaller Provisioning System" har också framtagits och utvärderats. Det ger administratören en plattform för att administrera OptiCaller klienter i en större skala och hanterar såväl installation samt inställningar av OptiCaller klienter på mobiltelefoner.

Acknowledgements

First, I want to express my thanks to my thesis examiner Professor Gerald Q. "Chip" Maguire Jr. He encouraged me from the first to the end of my thesis project. He also gave me valuable advise and suggestions. With respect to the research method, he told me to make progress gradually and write every day. Additionally, he provided me important technical suggestions for the project. Without his generous help, this project would not have come to reality. You have enhanced my capacity in more ways than you know; words are simply inadequate to express my sincere gratitude.

I am also extremely grateful for all the encouragement, support, and constructive advice from my industrial supervisor Jorgen Steijer. You supplied me with a good platform to display my ability and provided so many great thoughts about how to make the project useful in the real life. It is my pleasure to have done my thesis project in your company.

Additionally, I want to thank my colleagues, Li Zhang, Ioannis Metaxas, and Xiao Wu, for their support and constructive advice. It was a memorable experience for me to work with you.

Last but not least, I would like to thank my parents and girlfriend for their spiritual support and love.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

AP	Access Point
API	Application Programming Interface
APN	Access Point Name
APPARC	Application Architecture
CDMA	Code Division Multiple Access
CONE	Control Environment Hierarchy
DoD	Dial over Data
DTMF	Dual-Tone Multi-Frequency
EJB	Enterprise JavaBeans
FEP	Front-End Processor
GGSN	General GPRS Support Node r
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP over SSL
IDE	Integrated Development Environment
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IPC	Interprocess Communication
IP-PBX	Internet Protocol Private Branch Exchange
J2EE	Java Platform, Enterprise Edition
JME	Java Platform, Micro Edition
J2SE	Java Platform, Standard Edition
JAAS	Java Authentication and Authorization Service
JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
LDAP	Lightweight Directory Access Protocol
MEX	Mobile Extension
MIDP	Mobile Information Device Profile
MMS	Multimedia Message Service
OS	Operating System
OTA	Over-The-Air
PAM	Pluggable Authentication Module
PBX	Private Branch Exchange
PIPS	PIPS Is POSIX on Symbian OS
POSIX	Portable Operating System Interface for Unix
SDK	Software Development Kit

SGSN	Serving GPRS Support Node
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMS	Short Message Service
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SSL	Secure Socket Layer
TAN	Trunk Access Number
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TSY	Technology Systems
UDP	User Datagram Protocol
UI	User Interface
UID	Unique Identity
Uikon	User Interface Kernel on Display
UIQ	User Interface Quartz
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
WLAN	Wireless LAN
XML	Extensible Markup Language

1. Introduction

Nowadays, most companies pay the business phone bills of their employees, including their fixed line phones and mobile phone(s). The total amount of these phone bills in many companies is millions of Swedish krona per year. Especially for large companies whose employees make many international business trips, the phone bills for roaming cellular service can be very huge. Today the mobile roaming fee often comprises the majority of a business' phone costs.

"Dial over Data" (DoD) is a solution introduced by OptiCall Solutions AB. It is an integrated system which provides customers with a means to reduce the cost of their mobile phone calls. [1] There are three key components in the system: DoD clients, a DoD server, and a IP-PBX. Instead of dialing the callee's number directly, a request to call the callee is submitted to the DoD server over a data link by the DoD client. After the DoD server receives this request, it uses the IP-PBX to set up a call between the caller and callee. This call can use least cost routing techniques and can even utilize SIP calls to carry the signaling and traffic over an IP network. This cost saving solution is suitable for the common scenario in which the sum of the data fee (for the traffic to & from the client), the call receiving fee (at the client), the two call origination fees from the PBX, and the cost of the two calls themselves are less than the calling fee for the direct call. This saves a lot of money for customers especially when the customers are roaming abroad and there are one or more local gateways to the IP-PBX that can be used to turn a international roaming call into one or two local calls.

Besides cost savings, the DoD solution gives customers the ability to control and monitor the calling cost of their employees. For instance, using the DoD solution the IP-PBX can refuse to establish connections between employees, or monitor use of the company's subscription for private use.

Today, people rely on fixed line phones when they are in office and their mobile phones when they are away from the office. However, people work out of the office more frequently than before, so for convenience they use their mobile phones as a replacement for their fixed office phones. Unfortunately this can lead to high costs for the use of the cellular phone within the office. There are several ways to reduce the cost of these for example; turning calls into WLAN based calls, corporate site rate cellular calls, single number calling approaches, and the use of the DoD solution. Note that this thesis will not focus on costs savings or monitoring of employees calls, but rather will focus on the technical aspects of the client side of the DoD solution.

Call Through is a service for making cheap international calls, and already is provided by many telecommunications companies. Instead of making a direct call to the party the caller wants to speak with, the caller makes a call to the local call-through service access number. After the first call is established, the caller dials the actual callee's number. The service provider makes a call to the callee and bridges the two calls. The caller only needs to pay the local call and the service subscription fee instead of an expensive international call fee. Today, many companies provide such a call-through service to their employees to reduce the costs to the company for international calls.

Mobile extension is a popular concept which gives mobile users access to their corporate network's voice calling features on their mobile phones, for example features such as internal extension dialing, conferencing, and call transferring; while consolidating their mobile, fixed, and soft devices under a

single unique business phone number. [2] With a mobile extension system, users can use their mobile phones as a replacement for their fixed office phones. For an outgoing call, making a call looks and feels the same to the user no matter whether a call is made from a fixed office phone or a mobile phone. Depending on the PBX's configuration, even the callee cannot tell whether the source of the call is a fixed office phone or a mobile phone -- as the call always appears to be coming from the company's PBX. For an incoming call, both the fixed office phone and mobile phone can ring, thus users will not miss incoming calls to their office -- even if they are not in their office. Additionally, such a mobile extension can provide additional advanced functions, such as setting presence status, login groups, and transferring a call to another extension.

A DoD client was previously developed in Java. Although this client implements the DoD solution, However, this Java application is not sufficiently user-friendly. However, ease of use is especially important for telephony functionality. The reason for this lack of user friendliness is that to access a Java application, the user must normally open a Java application manager, launch the Java application, and interact with this application. This requires a change in the user's calling behavior -- as the user wishes to simply dial the number of the party that they wish to call or even more commonly click on a contact in the phone's phone book. Additionally, this Java application does not support mobile extension functionalities well.

The goal of this thesis project was to develop and evaluate a mobile extension application, called 'OptiCaller', running on a mobile phone running the Symbian operating system (OS). The most significant advantage of the OptiCaller application is transparency. In this approach the OptiCaller runs on a Symbian OS based mobile phone in the background. This approach enables users to make a call in lower-cost way **without** changing their behavior -- as the application intercepts the user's dialing interaction with the phone as it is dialing a number or choosing a contact in the phone's phone book. Additionally, OptiCaller supports making call-through calls and a number of mobile extension functions; including setting presence status, ongoing call services, and so on.

The new OptiCaller application will be evaluated from several aspects, including call performance as compared to the existing Java application (which is the new Java client, see Chapter 2) and power consumption. These two metrics were chosen as the user wants to simply make calls, but with an application running in the background they might be concerned about the power consumption of such an application - to avoid significantly reduced talk and standby time.

Additionally, a provisioning system called 'OptiCaller Provisioning System' was designed and implemented in order to handle the provisioning of the OptiCaller. By using the provisioning system, the OptiCaller application can be delivered and easily managed. This provisioning system allows over-the-air (OTA) installation [3] and remote updates the client application's configuration. This provisioning system will be evaluated in terms of its load capacity. Specifically, we want to determine how the load on the provisioning system will scale with the number of users.

Chapter 2 introduces related work. Chapters 3 to 9 describe the background of this project. The material about the Symbian OS and its C++ APIs are the subject of Chapter 3. Chapter 4 describes the Symbian OS architecture, while Chapters 5 to 7 explain the most important three layers of the Symbian OS in detail, specifically the User Interface Framework Layer, Application Services Layer, and OS Service Layer (they are described in this order). Chapter 8 introduces the Java Platform, Enterprise

Edition (J2EE) and the application server used in this project. Chapter 9 explains how to send an SMS from a PC in a program. OptiCaller is described in Chapter 10, while the OptiCaller Provisioning System is described in Chapter 11. Chapter 12 focuses on the evaluation about the OptiCaller and the OptiCaller Provisioning System. Chapter 13 gives a summary of our conclusions and suggests future work.

2. Related Work

Currently, there are two ways for a user to access the DoD solution: (1) via a Web browser or (2) via a mobile phone Java application. [4] The web interface is designed for web browsers such as Mozilla FireFox and Internet Explorer. Additionally, a mobile client developed in Java Platform Micro Edition (JME) was previously designed and implemented for mobile phones supporting the Java Mobile Information Device Profile 2.0 (MIDP provides a standard Java runtime environment [5]). Figure 1 and Figure 2 show the two user interfaces currently available.

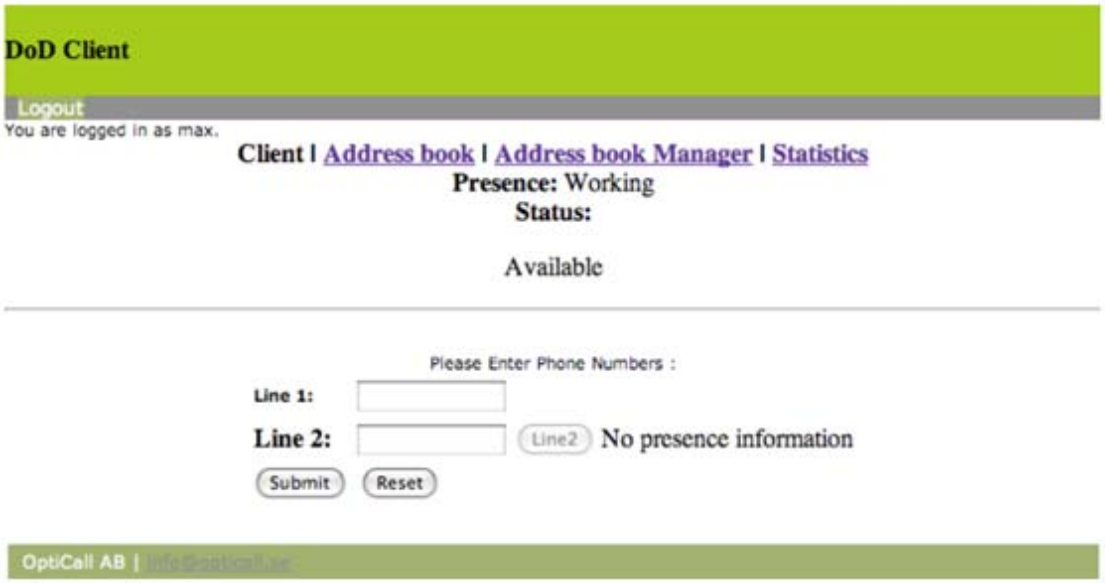

Figure 1: DoD Web User Interface

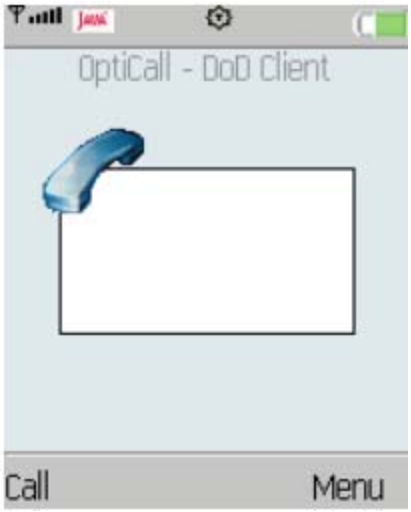

Figure 2: DoD Java User Interface

Both the web client and mobile client support call-back. Call-back is one of the two basic methods used to realize the DoD solution. Rather than directly calling the callee, the caller sends a data request to the DoD server, the server causes a PBX to establish calls to both caller and callee, then bridges these two

calls together to create a connection between the caller and callee. Hence the origin of the name: "call-back".

However, both clients suffer from a number of problems. A disadvantage of both clients is that they require that the human change their usual dialing behavior. For instance, the Java client requires the caller to perform 4 steps to make a call:

1. Open the Java application
2. Input the callee's number in the application user interface
3. Send the request and wait for an incoming call-back call
4. Accept the incoming call in order to connect to the other party

Additionally, the two current clients do not support call-through (another method that can be used in the DoD solution) nor do they support mobile extension functions such as presence setting and ongoing call services. For details of the call-back and call through solutions as seen by the server, see the companion master's thesis by Zhang Li. [6] It is worth mentioning that a new Java client is developed in another thesis project by Ioannis Metaxas. [7] Figure 3 shows the interface of the new Java client.

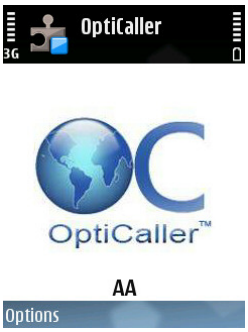

Figure 3: New Java Client User Interface

To increase the ease of use of the DoD solution, a more integrated client needs to be developed. In this thesis project, a client tailored for a Symbian OS based phone was designed, developed, and evaluated. One goal was to explore alternative ways for user dialing **without** changing the user's dialing behavior (which could be manually dialing number or choosing a contact from the phone's phone book). Another goal was to support the mobile extension functions.

There are other cost-saving products in the market. A solution using a Skype PBX [8] was introduced by OnState [9]. Users make a call into the Skype PBX, then the call will forwarded onward via Skype. This solution is similar to call-through and is one of functions provided in the new OptiCaller application. In the call-through approach the user makes a voice call into the IP-PBX, enters the phone number of the callee (for example using DTMF), and the IP-PBX will use a SIP trunk to make a call to the callee. Additional details of call-through are given in following chapter. Additionally, TelePo [10] provides a solution called Telepo Mobile+ [11]. It runs as a background application providing automatic access to the enterprise voice network using local access gateways or automatic call-back. [11] Compared to Telepo Mobile+, the OptiCaller supports triggering a call-back via a data-link; which takes full advantage of the wide use of GPRS and 3G. It also supports SMS call-back. Additionally, OptiCaller is compatible with most PBXs supplying mobile extension functions, because instead of being tailoring to a specific PBX, OptiCaller is designed in a flexible way. OptiCaller allows users to edit their mobile extension function codes by themselves (or the OptiCaller Provisioning System) in order to adapt to

different PBX solutions. In contrast the Telepo Mobile+ is tightly integrated with their choice of a SIP proxy server.

3. Symbian OS C++

The Symbian OS was developed in a non-standard C++. There are some differences from the standard C++, these differences are described in the following sections.

3.1 Basic Types

It is possible to use the basic types of standard C++, but it is recommend that a programmer use the Symbian specific types to preserve compiler independency. Table 1 shows some of the corresponding data types of Symbian OS C++ and standard C++.

Table 1: Basic Data Type in Symbian OS and Standard C++

Symbian OS C++ basic data type	Standard C++ basic data type
TUint	uint
TInt	int
TReal	float
TBool	bool
TAny	void

3.2 Naming Convention

Symbian OS uses a naming convention to indicate what is important and to make the source code more readable. The naming convention rules in Symbian OS C++ are [12]:

For a class name, there are four main prefixes for different types of classes:

T T classes are built-in classes which do not need deconstructor. Examples are: TInt, TFileName, and TBuf.

C C classes are any classes derived from CBase class. They always are allocated on the heap. Examples are: CActive, CCoeControl, and CAknView.

R R classes own the resources. Most of them need to be connected to or open the resources before using them. And they need to be closed at the end. Examples are: RFile, RTimer, and RWindow.

M M classes are interfaces. Symbian OS supports multiple inheritances by them. The visual functions of M classes need to be implemented. Examples are: MEikMenuObserver, MHTTPDataSupplier, and MProgressDialogCallback.

For data names, there are four main prefixes for different types of data:

- Arguments use the prefix ‘a’.
- Class member data uses the prefix ‘i’.
- Constants use the prefix ‘K’.
- Enumerated constants use the prefix ‘E’.

For function names, the convention is the same as standard C++, except for the leaving function. More details about the leaving function are given in the following section.

3.3 Exception Handle

Unlike standard C++, Symbian OS C++ has its own mechanism to handle exceptions. Some new concepts are introduced such as the Leaving Function, CleanupStack, and Two-phase Construction. [13]

3.3.1 Leaving Function

Lack of resources is a common environment error for programs; especially for the Symbian OS, since mobile devices frequently have more limited resources than larger devices such as PCs and laptop computers. Lack of memory is the most common issue. If several functions ask for resources at a critical time, when there are insufficient resources available, then the function will leave, i.e., be terminated. Functions which may leave are called leaving functions. This behavior is similar to the functions that may throw exceptions in standard C++.

It is important to handle the leaving functions properly since failing to do so could lead to serious problems. For instance, consider a leaving function that applies for memory twice within the function. Assume that it receives memory successfully at the first request, but it fails to receive the requested memory allocation with its second request. Due to the second request failing, the function leaves, but the memory allocated the first time is still allocated leading to a memory leak. Such memory leaks can be crucial for a mobile device since the resources are so limited. Restarting the device could release the allocated memory, but if the allocated memory is in the persistent storage, it is unavailable forever. This means that all functions which allocate resources must be carefully designed so that upon exiting they properly return resources that should returned to their respective resource pools. This is addressed using Symbian OS's exception handling - as described in the next subsection.

3.3.2 CleanupStack and TRAP/TRAPD

Symbian OS uses two mechanisms, CleanupStack and TRAP/TRAPD, to handle exceptions. TRAP/TRAPD is similar to try and catch in standard C++. TRAP/TRAPD runs a leaving function, and puts the leave code into an integer. If the code is KErrNone, this indicates that the function executed successfully. Otherwise, the program needs to correct the situation indicated by the error code. The difference between TRAP and TRAPD is that an error variable needs to be defined for storing the error code before TRAP is used; while TRAPD can be used without an error variable definition.

The CleanupStack works with TRAP/TRAPD to prevent memory leaks when a leave occurs. It performs three operations:

- It stores the address of allocated memory, pushing it onto a stack.
- After finishing the operation which may leave, it pops the address out of stack.
- TRAP/TRAPD marks when the leaving function starts. If a leave occurs, the memory blocks allocated after that mark are released, and the allocated address pointers (to these blocks) on stack are popped.

The name convention for leaving functions is:

- Leaving function uses the suffix ‘L’.
- If a leaving function has the suffix ‘LC’, this means that it pushed one item onto the CleanupStack.
- If a leaving function has the suffix ‘LD’, this means that it deletes or releases the item which is pushed onto CleanupStack.

3.3.3 Two-phase Construction

Two-phase construction is a typical use of CleanupStack and TRAP/TRAPD. The Symbian OS introduced two-phase construction to avoid memory leaks when constructing an object.

The construction is divided into two steps. The whole construction is TRAPed by a framework. All safe operations such as standard construction functions will be executed in the first step. All potential leaving operations are executed in the second step. The object’s address will be pushed onto the CleanupStack before the construction moves to the second step. Thus, if the second phase of the construction leaves, the framework will release the memory already allocated and pop the object pointer off of the stack.

3.4 Active Objects

Active objects are a fundamental part of the Symbian OS. In an operating system, a service can be performed either synchronously or asynchronously. A synchronous request means the control does not return to the caller until the service is completed. In contrast, an asynchronous request means that control returns to the caller immediately after submitting the request. The service could complete sometime later.

Normally, operating systems use multi-threading and multiprocessing to deal with multiple tasks. The Symbian OS supports multi-threading. Threads are preemptively executed by the kernel which controls thread scheduling. The thread with the highest priority which is not in the blocked state is run by the kernel. A blocked thread is waiting for an event to happen, for example a service completion. A context switch occurs if the current thread is suspended. [14]

However, mobile phones are not as powerful as desktop PCs. Hence multi-threading should be avoided on mobile phones due to the limited resources (specifically computation power and memory). To provide a light weight alternative to multi-threading, the Symbian OS introduces Active Objects to deal with multi-tasking.

Active objects are a unique concept to Symbian OS. It is a way to implement non-preemptive multi-tasking inside a thread. The core of an active object is active scheduler which is similar to a mini-kernel for each thread. There is only one active scheduler in each thread. It manages several active objects. Each active object is responsible for making a request and handling its completion. A single thread usually issues many outstanding requests. The Active scheduler handles the completion of requests and calls the corresponding active object to handle the request. Each active object has a RunL( ) method which is called when the request of this active object completes. As a kernel managing multiple threads, if several requests complete at the same time, then the active scheduler decides which active object corresponding to the completed request is called. [15]

Active objects are widely used in the Symbian OS. For example, the Graphic User Interface (GUI) framework is a typical user of active objects. Usually, developers only need to implement the OfferKeyEventL( ) method to handle a keyboard event. The method is invoked in the RunL( ) method which is implemented by the GUI framework. Symbian applications usually only have one thread, so that all asynchronous requests are handled by active objects.

As we known, time and memory consumption are two key factors for software applications. According to the test performed by Aapo Haapanen [16], a thread-based solution consumes about ten times as much memory as the active-objects-based solution. Additionally, an active-objects-based solution is much faster than a thread-based solution. Therefore, using active objects instead of multi-threading improves the performance of Symbian OS applications.

3.5 Client-Server Model

The Symbian OS offers several services at different layers. Normally, the server runs in a separate thread; while processes are used to execute the application process. Therefore, the Symbian OS supports interprocess communication (IPC) mechanisms. [12] Event driven processing is the base of the Symbian OS. Most event interactions are initiated by the user. Thus a Client-Server model using the IPC mechanism is a very common approach to implementing services running on the Symbian OS. Most of the system services in Symbian OS are Client-Server-based; such as the Window Server, File Server, and Telephony Server.

Figure 4 illustrates this Client-Server IPC mechanism. Firstly, the client needs to establish a session with the target server. This session is used for all further communication between the client and the server. There are two kinds of messages in such communication: client requests and server responses. The Symbian OS kernel manages the sessions. For each session, the kernel delivers the request to the appropriate server. The server retrieves the data in the request (if there is additional data), then processes the request and notifies the client that its request is completed. Since this is connection-oriented session-based communication, the Client-Server IPC mechanism is a guaranteed request-completion mechanism. [12]

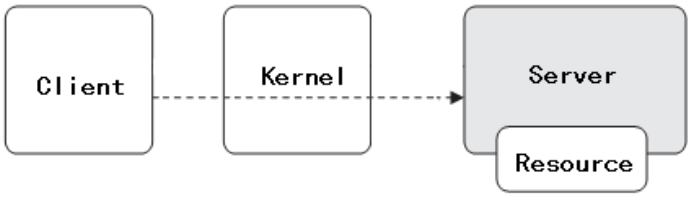

Figure 4: Client-Server IPC mechanism

The Client-Server IPC mechanism has advantages in terms of efficiency and security. [17] Multiple clients can use a single server at the same time, while these clients and the server are executed in different processes. All the session management is done by the kernel. Therefore, a client failure will not affect the server. This Client-Server IPC mechanism supports both synchronous and asynchronous requests.

However, in such a Client-Server IPC mechanism, each client must know which server provides the service it wants. Additionally, requests may be suspended by the kernel making this approach unsuitable for real-time communication. The Symbian OS provides two additional IPC mechanisms for interprocess

communication; a Publish and Subscribe IPC mechanism and a Message Queues IPC mechanism. [12] Since in this thesis project the server needed is known and there is no real-time requirement, these other two IPC mechanisms are not used.

4. Symbian Architecture Overview

The Symbian operating system can be represented as a layered model (see Figure 5). This model divides the Symbian OS architecture into five layers, from top to bottom: User Interface (UI) Framework, Application Services, Operating System (OS) Services, Base Services, and Kernel Services & Hardware Interface layers. [18] The Symbian OS also supports Java ME on top of the OS Services layer.

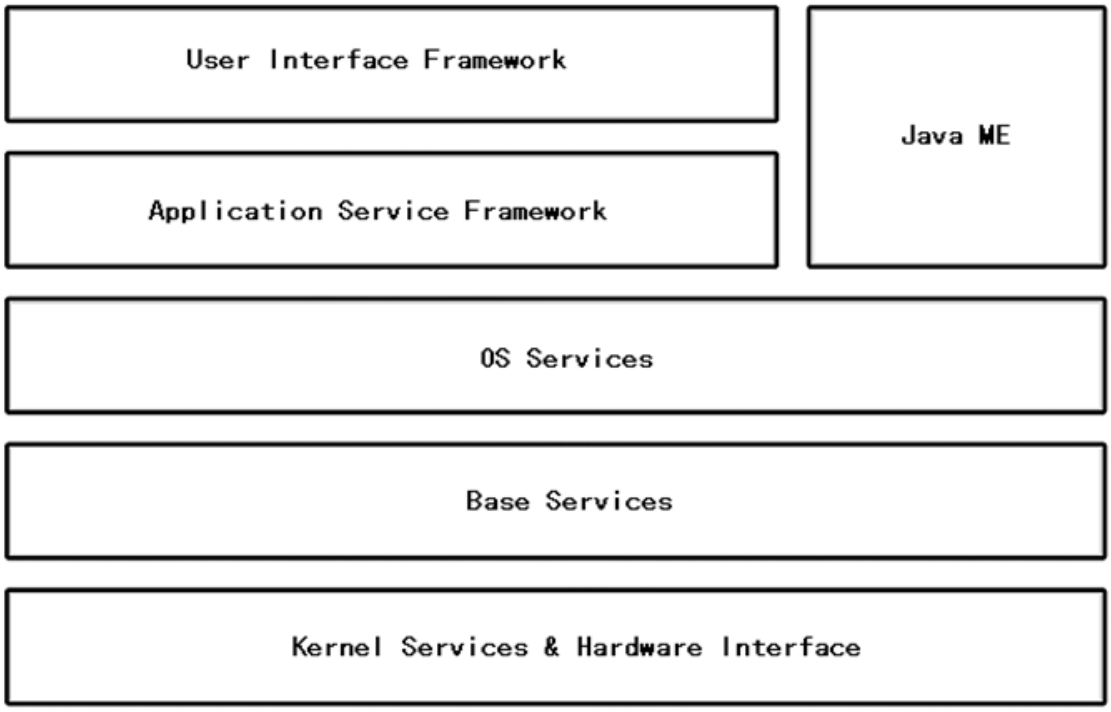

Figure 5: Symbian OS layer model [18]

4.1 User Interface Framework Layer

The user interface (UI) Framework layer is the top most layer in the Symbian OS layer Model. It delegates tasks to the Application Services layer. This UI framework layer is used for constructing a user interface by providing UI frameworks and libraries.

The basic class hierarchies provided by the UI Framework layer implement user interface controls. These class hierarchies include the Uikon framework, the control environment hierarchy (CONE), and the front-end processor framework. [19] The classes are also available to applications besides the user interface. Uikon and CONE are the most important classes for the GUI, since together they provide the framework which defines the basic GUI behavior.

The UI Framework layer also provides other frameworks and utilities, including the UI Graphic Utilities and Graphics Effects components, the Animation components, and the Grid framework. Together they support fonts, colors, and graphics effects for the user interface. Some of these frameworks and utilities are available to applications besides the user interface. More details are given in Chapter 5.

4.2 Application Services Layer

The Application Services layer provides services for applications. These services can be divided into three categories: system-level services, technology-specific logic, and specific individual applications. [18]

System-level services include basic application frameworks which can be used by all applications. As an example, MIME-based content recognition and handling can recognize files according to their MIME type so that a suitable application can be launched to open that file.

Technology-specific logic supports several application-level standards, such as email standards (including SMTP, POP, and IMAP), internet data protocols (including WAP, HTML, and HTTP), and phone message standards (including SMS and MMS).

Specific individual applications include personal information management, device management and client provisioning, not only on-device but also over-the-air (OTA).

Most of services supplied by this layer are used for standard phone functionality. However, they are also available to the third party developing new applications.

More details of this layer, as related to this thesis project, are given in Chapter 6.

4.3 OS Services Layer

The OS Service layer supports graphics, communications, multimedia, and other system-level utilities. It also provides a number of generic system frameworks and libraries such as the C Standard Library. It extends the bare system into a complete, programmable operating system.

This layer can be divided into four major blocks: [18]

- Generic OS Services
- Multimedia and Graphics Service
- Connectivity Services
- Communications Services

These blocks are not isolated, but in fact are related to each other. For instance, Generic OS Services is used by other blocks, and Communications Services is used by Connectivity Services.

These services seldom are used by applications directly. Instead, applications use the service by way of higher-level frameworks. However, some core services are used by all applications, such as the window services (no matter whether the applications knows it or not).

More details, as related to this thesis, are given in Chapter 7.

4.4 Base Services Layer and Kernel Services & Hardware Interface Layer

The Base Services layer and Kernel Services and Hard ware Interface layer are the lowest layers of the Symbian OS. These two layers comprise the operating system kernel, device drivers, and the device-driver framework support. [18]

The Base Services layer provides basic operating system services. These basic services include file services, persistent storage services, and so on.

The Kernel Services and Hardware Interface layer is the lowest layer of the Symbian OS. It consists of the kernel and the infrastructure needed to boot and run the kernel on top of a specific hardware platform. It is responsible for fundamental operating system services, including initializing hardware accessing devices, creating and managing system kernel abstractions such as threads, processes and memory spaces, and so on.

These two layers constitute a minimal operating system which can boot and run other code.

5. User Interface Framework layer

5.1 User Interface Platforms

Nowadays, there are a lot of mobile phone manufactures whose products are based on the Symbian OS. Although the basic operating system is the same, these mobile phones have their own distinct looks and behaviors. The reason for this is that the different mobile phone manufacturers have implemented their own User Interface Platforms on top of the UI Framework layer.

When the Symbian OS is delivered to licensees, a minimal test user interface is included. This is an incomplete user interface which is called 'TechView'. It is used in the emulator (when running on a Microsoft OS platform) to provide a Microsoft Windows-based implementation of the Symbian OS. Mobile phone manufactures that have a Symbian OS license either develop their own user interfaces or license/buy a suitable one.

Currently, there are three main UI platforms used in the market: S60, UIQ, and MOAP. [19] S60 was developed and is licensed by Nokia to other vendors. Some of these manufacturers who license and ship S60 phones based on Symbian OS are: Lenovo, Panasonic, and Samsung. UIQ is owned by Sony Ericsson. Other manufacturers (such as Motorola) license and ship UIQ phones. MOAP was developed by the FOMA consortium in Japan and is licensed and shipped by Fujitsu, Mitsubishi, and Sony Ericsson. S60 and UIQ are two most popular UI platforms now. The UI Framework layer has some differences corresponding to which UI platform is built on top of it. We will not discuss the difference in this report since this project only focused on S60 as it is the most common UI platform.

5.2 User Interface Framework Layer

The UI Framework layer underlies the various UI platforms supplied by mobile phone manufacturers. It provides frameworks and libraries for developers to construct a customized user interface on top of the Symbian OS. For instance, it has frameworks for extending a customized user interface and provides some generic frameworks such as animation.

Although the UI Framework layer has become thinner since the since the introduction of the UI platforms on top of it have developed rich user-interface functionality. However, the UI Framework layer sill provides important core user-interface functionality which determines basic application behavior, such as window interactions. This is achieved by two major components: the Uikon framework and the Control Environment (CONE). (See Section 4.1)

5.2.1 The Control Environment

Controls are the basic concept used in CONE. Controls are window-using, possibly nested, rectangular screen areas that accept user input and other events. [12] These events include redraw, user-input, foreground-focus events, and so on. Some events are supplied by the Window Server to CONE, including user-input events, while some events are generated by controls themselves such as the change of focus event.

CONE defines the base classes that encapsulate basic behavior, such as user input, and the relationship between controls and their environment. CONE could be seen as an abstract middle layer between the Window Server providing low-level functionality and the concrete user-interface classes provided by Uikon (see Section 5.2.2). The events supplied by the Window Server to CONE are routed to controls.

There are three essential classes defined by CONE: CCoeControl, CCoeEnv, and CCoeAppUi.

CCoeControl is the base class from which all other controls are derived. It displays the application data on the screen and allows users to interact with the application. This is sometimes called a 'view'.

CCoeEnv is the class which encapsulates the application session with the servers including the Window Server, File Server, and so on. Every application owns a single object of a class derived from CEikonEnv, which is a sub-class of CCoeControl. This object is responsible for routing input-event messages from the Window Server to the application framework CEikAppUi class.

CONE also defines the application user interface base class CCoeAppUi. Handling commands is the main responsibility of an application UI class. These commands are specified in the application's resource file. The application is also responsible for dealing with other kinds of events sent by the OS to the application, including key events, application switching to foreground event, etc. CEikAppUi is an application UI class derived from CCoeAppUi class. An application always owns a concrete application UI class derived from CEikAppUi. Some UI-specific features may be added to this application UI derived class.

5.2.2 The Uikon Framework

Uikon stands for User Interface Kernel on Display. It is the generic Symbian user interface on top of which various manufacturer specific user interfaces are implemented.

Uikon is based on two important frameworks: the Control Environment (CONE) and the application architecture (APPARC) which is a framework for applications and application data. Figure 6 illustrates the Uikon Framework. [12]

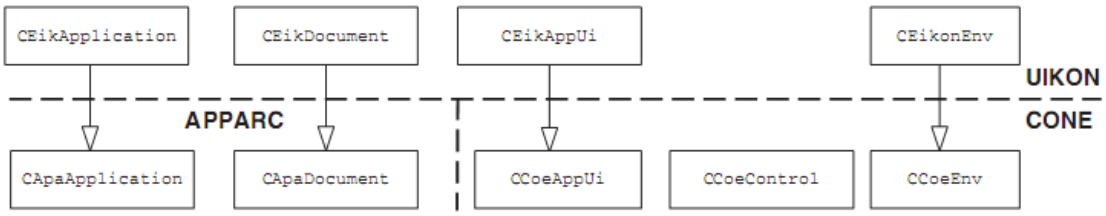

Figure 6: Frameworks on which UIKON is built [12]

APPARC consists of application class (CEikApplication) and document class (CEikDocument). Each application always owns a pair of concrete derived classes from the application class and document class.

The application class (CEikApplication) defines properties of the application, such as its globally unique ID (UID) which is defined in project's '.mmp' file. This class is responsible for creating the document-derived class. This derived class is the interface to the resource file and the document.

The document class (CEikDocument) represents the data model for the application. In file-based applications, it stores and restores the application's data. Even if the application is not file-based, a

document class must exist in each application for storing and restoring the application data. Additionally, it creates an application UI class which is a component of CONE.

The framework also includes a component named look-and-feel which is provided by each user interface variant. This component defines a standard set of methods which the user interface variant can use to define the different behaviors of user interface elements. The different behaviors include the layout of windows; system fonts; and the appearance of the toolbar, dialog, and button; and so on. Figure 7 shows the different behaviors for an alert dialog on S60 and UIQ (respectively).

In addition, there is a small component called the Uikon Error Resolver Plug-in. This component is used by the UI variant to map system error codes to strings. This mapping makes the system errors more understandable to users.

Figure 7: Alert Dialog of S60 (left) and UIQ (right)

5.2.3 The S60 View Architecture

Avkon provides the S60-specific application framework layer implemented on top of Uikon. It adds further UI libraries that can include controls derived from the ones provided by Uikon. [20] Moreover, it defines many S60-specific controls, such as list box, editor, and dialog. Table 2 indicates the generic Symbian OS application framework classes, provided by S60. [12]

The traditional S60 platform applications follow the traditional Symbian OS UI application architecture, which is illustrated in Figure 8. [21] As shown in this figure, the views are derived from the standard CCoeControl. These views are responsible for showing data and accepting user interaction events.

The application UI class is derived from CAknAppUi which is called a UI controller. This derived class creates one or more CCoeControl-derived classes, enables them to handle key events, and switches between the CCoeControl-derived classes. This application UI class also handles menu commands; and receives and handles events such as when an application is brought to foreground from the background in the run-time environment.

However, using a traditional UI architecture, the developer has to do all the view management (such as view switching). Sometimes this will be very complicated. Therefore the view architecture was

introduced to simplify development. Figure 9 shows the S60 view architecture of S60. [21] As in a traditional UI architecture, the views are derived from the standard CCoeControl.

Table 2: Application Framework Classes [12]

Class	Generic Symbian	S60 (Avkon)
Application	CEikApplication	CAknApplication
Document	CEikDocument	CAknDocument
Application UI	CEikAppUi	CAknAppUi
View	CCoeControl	CCoeControl

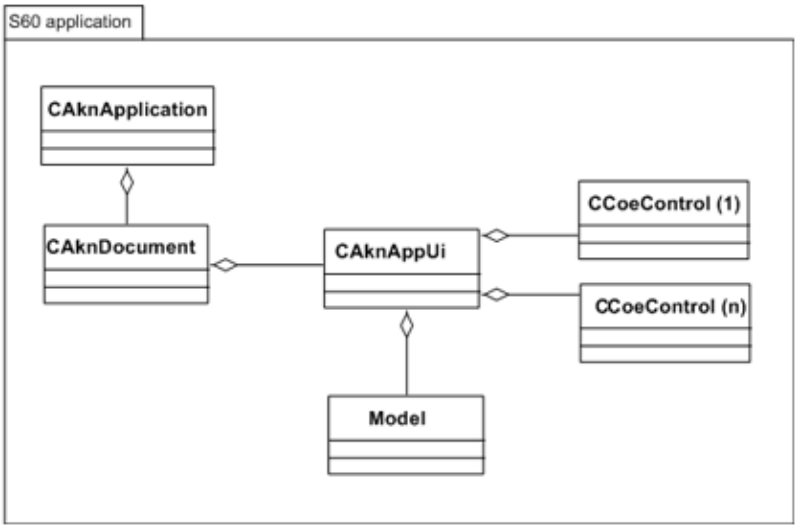

Figure 8: Tranditional Symbian OS UI Application Architecture Classes [21]

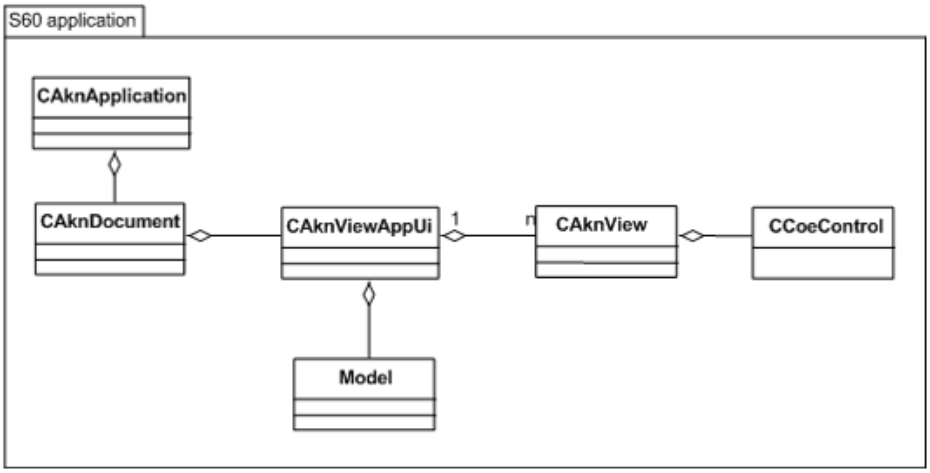

Figure 9: S60 View Architecture Classes [21]

A new concept is introduced by the S60 view architecture is the view controller (CAknView). It owns one or more CCoeControl-derived objects. It implements the MCoeView interface which manages the view ID by which the corresponding view can be activated. It also deals with the events which will be trigged when a view is activated or deactivated. For instance, when the view controller is activated, the view controller could decide which view (CCoeControl-derived class) should be activated. When a view is activated, the view controller adds it to the control stack, which means that the activated control is

registered to receive key events. The view controller also has the ability to handle the command which is a partial responsibility of the application UI class. Using the view controller, views can be easily switched by simply invoking one method.

The application UI class in the S60 view architecture is **not** derived from the standard CAknAppUi directly; rather it is derived from CAknViewAppUi. It creates one or more view controllers and manages them; including adding or removing the view controllers into or from the view framework. It switches the view controllers and handles the commands passed to it by the view controllers. It also handles the events which are not handled by the view controllers, such as the event when an application is brought to foreground event.

The disadvantage of the S60 view architecture is that it has more limitations than the traditional UI architecture. For instance, it cannot provide views that could be used in other applications nor can it be layered over other applications.

5.2.4 The Front-End Processor framework

The Front-End Processor framework (FEP) is another interesting framework provided by the UI Framework layer. It provides abstractions that implement user-input capture and preprocessing. [12] It is managed by CONE -- which creates, owns, and destroys the FEPs.

The FEP is based on the CCoeFep class which owns a high-priority and invisible control thread. Because the FEP has a high-priority, it receives keyboard events **before** other controls. As a result FEP captures and preprocesses the input events, then returns them to the control stack as new events for lower-priority controls. For instance, when a user uses the visual keyboard to input a character '1', the FEP captures the event, then returns a key event whose code is '1' to the control stack. Subsequently the other controls in the control stack can receive this event and handle it.

6. Application Services Layer

The Application Services layer provides services for applications. As stated earlier, it supports email standards (such as SMTP, IMAP, and POP), phone-messaging standards (such as SMS and MMS), and internet document and data protocols (such as HTTP, XML, and HTML). For this thesis project, HTTP support will be used; hence the rest of this section will focus on the Application Services Layer HTTP framework.

6.1 HTTP Framework

HTTP uses TCP as its transport protocol. It uses a client-server model. The Symbian OS supplies a HTTP framework to support all HTTP-defined request methods, including GET and POST. It provides a set of application programming interfaces (APIs) which helps programmers to develop applications without needing to pay attention to details of the HTTP stack. The framework also supports HTTPS as well, for added security (see section 7.2.2).

To develop an HTTP application, a HTTP session needs to be established using the API called RHttpSession. Within a session, a client sends a request to and waits for a response from an HTTP servers. Since creating a session consumes time and memory, it is recommended that only one session is created in an application. Fortunately, multiple HTTP transactions can take place during one session. Several different session properties can be set, for instance, the network connection could be set for the session so that a user prompt will not be triggered to ask the user to select the network connection to be used. The ownership of this connection belongs to the application, thus after finishing using the connection it is the application's responsibility to close the network connection - rather than this being a responsibility of the framework. The session properties apply to all transactions within the session. However, some properties could be set for a specific transaction overriding the session's settings.

The processing of an HTTP transaction is encapsulated in an API called RHTTPTransaction. It creates the request message, including the header and the body, and submits the message to the HTTP server.

To customize the request body, an object which implements the interface MHttpDataSupplier is needed. This object supports creating the body in parts if the data is large.

The HTTP framework provides a way to monitor the transaction so that the developer can concentrate on dealing with processing the data. To monitor a transaction, the interface MTransactionCallback needs to be implemented. Table 3 shows the common events supported by the HTTP framework. [22]

Table 3: HTTP Transaction Event Code [22]

Event Name	Description
EGotResponseHeaders	Indicates that the response has been received and the status line and header field information can be retrieved
EGotResponseBodyData	Indicates that body data is ready for access
EResponseComplete	Indicates it is the end of the body
ESucceeded	Transaction completed OK, the session needs to be closed
EFailed	The cause of the failure needs to be investigated

StringPoll [22] is an important concept for the HTTP framework. It efficiently deals with standard strings. This is particularly useful for HTTP, because HTTP uses a lot of well-know and standard strings such as the header field name. There is member data in the RHttpSession class called stringpool into which the commonly used HTTP strings will be loaded. They can be conveniently passed to an HTTP API. For instance, when the request method needs to be defined as POST, the value stringPool.StringF(HTTP::EPOST,RHTTPSession::GetTable( )) can be directly assigned to.

7. OS Services Layer

Except for the kernel server and file server, all core services are provided in the OS Services layer. The services can be classified into four classes: Generic OS Services, Multimedia and Graphics Services, Connectivity Services, and Communications Services. [18] Graphics Services provide an efficient architecture to support various devices. Meanwhile, Multimedia Services provide a framework for supporting multimedia such as audio, video, and cameras. Connectivity Services provide frameworks for supporting device-host connectivity functions. With this, mobile phones can communicate with a host platform, for example a desktop, for data backup, remote software installation, and so on.

The following sections will introduce the Generic OS Services and Communications Services which are most relevant for this thesis project.

7.1 Generic OS Services

Generic OS Services include a number of general services and some useful frameworks and libraries. The following subsections will highlight some of these (that are most relevant to this thesis project).

7.1.1 Task Scheduler

The Task Scheduler is one of most important services provided in the OS Services layer. It is an application-launching server which takes responsibility for creating, querying, and editing time-and-condition-based tasks. In other words, the application is invoked by the task scheduler when a specific task trigger occurs. This task schedule determines if the time is up or a condition is met. For instance, OptiCaller is configured to launch when the cellular phone starts-up; this behavior is controlled by the Task Scheduler.

7.1.2 Event Logger

The Event Logger provides a service for logging and filtering system events for the system or applications. Maintaining a list of recent calls is a typical use of the Event Logger. Events expire when their lifetime ends.

7.1.3 Certificate and Key Management Framework

The Certificate and Key Management framework provides a complete framework for managing certificates and keys. This framework supports public key cryptography for RSA, digital signature algorithm, and Diffie-Hellman key pairs, assignment of trust status and certificate-chain construction, and key validation and revocation. [18][23] This framework is used by system clients and licensee applications, such as the Application Installer, browsers, and virtual private network client applications.

The private public key infrastructure keys which are used to sign data and verify signatures are stored in the Key Store. The framework provides APIs for storing and retrieving keys. Similarly, the certificates which belong to the root and users are stored in a Certificate Store. However, generating the certificate/key pairs is not supported by Symbian frameworks. The certificate/key pairs can be provided by the trusty organizations, such as VeriSign [24] and TrustCenter [25]. In this project, the

certificate/key pair is used for signing the OptiCaller installation file. Because the application is signed, the application can utilize protected capabilities, such as accessing the mobile phone network without requiring the user to explicitly give their approval each time. [26]

7.1.4 C Standard Library

The C Standard Library provides a basic subset of the standard ANSI C library functions and POSIX system calls. [18] POSIX defines APIs for software compatible with variants of Unix OS. The Symbian OS supplies a library called PIPS which stands for 'PIPS Is POSIX on Symbian OS'. This library facilitates porting applications from other platforms to Symbian OS based mobile phones.

However, the C Standard Library does not support some Symbian OS native idioms such as the active object which the Symbian OS uses for achieving multi-tasking without multi-threads. More details about active object are in Section 3.4. More details about Symbian OS C++ are in Chapter 3.

7.2 Communications Services

The Symbian OS supports various communication technologies including telephony services, IP network services, USB, Bluetooth, and so on. [18] The three main communication services used in this thesis project are: telephony services, IP network services, and messaging services.

7.2.1 Telephony Services

The Telephony Services are provided by the ETel Telephony Server. It supplies a number of telephony APIs by which application can access a generic phone link. Access to the phone link can be divided into two categories: data-centric and voice-centric. Data-centric applications do not use the ETel Telephony Server directly; rather they use higher-level APIs such as the ESOCK APIs. Interaction with the ETel Telephony Server is done via the Network subsystem which provides Networking Services. In contrast, voice-centric applications directly use the ETel Telephony Server. These services include dialing an outgoing call and answering an incoming call. Both of these operations are of particular importance to this thesis project.

The ETel Telephony Server provides a client-side API to implement the Symbian OS client-server framework. [27] It supports sending a request to the telephony stack for dialing or answering a call. As a standard client-server framework, clients need to open a session with the ETel Telephony Server; then they can open sub-sessions with phone, line, and call objects. The phone object provides the ability to access the specific phone, for example, to get the status of the phone hardware. It is implemented in the class RPhone. The line object provides the ability to access a specific line, for example, to get the number of calls opened from this line. It is implemented in the class RLine. The call object provides the ability to access functions related to a specific call, such as answering an incoming call. It is implemented in the class RCall.

A Technology SYstems (TSY) module is a plug-in which could be seen as a middle-layer between ETel Telephony Server and the telephony stack. Before the client requests arrive at the telephony stack, they should be translated into appropriate requests by TSY module. Symbian OS provides four TSYs as reference implementations. They are Multimode TSY, Global System for Mobile communications (GSM) TSY, Code Division Multiple Access (CDMA) TSY, and SIM TSY. It is worth noting that the

Multimode TSY supports both GSM and GPRS functionality. It uses Hayes AT style commands to communicate with the telephony stack. SIM TSY is a simulator module and does not communicate with actual hardware. The TSY should be loaded before trying to use the functionality provided by ETel Telephony Server. Since a phone may support several TSY modules, it is recommended that the name of the current TSY module should be obtained by querying the communications database.

The main two functions supplied by the ETel Telephony Server are answering an incoming call and dialing an outgoing call. To achieve these, ETel Telephony Server provides simple APIs. Additionally this server provides services such as obtaining an IMEI or IMSI; and retrieving information about the connected network, for instance the connected Cell ID, and monitoring call and voice line status. Table 4 [28] lists the ETel Telephony Server functions that are relevant to this thesis project and Table 5 [28] shows the call status values relevant to this thesis project.

Table 4: Part of ETel Telephony Server functions [28]

Functions	Description
CTelephony::DialNewCall( )	Initiate a new call
CTelephony::AnswerIncomingCall( )	Answers an incoming new voice call
CTelephony::SendDTMFTones( )	Transmit DTMF tones across the current active voice call
CTelephony::NotifyChange( )	Register the events on this line including incoming call events and outgoing call events

Table 5: Part of Call Status [28]

Call Status	Description
EStatusUnknown	Indicates that the status is unknown
EStatusIdle	Indicates that no active calls
EStatusRinging	Call ringing status
EStatusAnswering	Call answering status
EStatusDialling	Call dialing status
EStatusConnecting	Call connecting status
EStatusConnected	Call connected status
EStatusDisconnecting	Call disconnecting status

7.2.2 Networking Services

As mentioned before, Symbian provides a HTTP framework to support HTTP transactions. These HTTP transactions are based on using a TCP/IP network stack. This network stack is the core of the implementation of Networking Services. The networking services also include network security protocols and packet-data services such as GPRS, UMTS, and Wi-Fi.

The Network Security protocols work at different levels in the networking stack. TLS and SSL operate at the transport level, providing encryption and decryption per TCP segment. Symbian supports TLS v1.0 and SSL v3.0, as they are used in HTTPS and SyncML. IPSec operates at the network level. It is used for support secure networks, for instance virtual private networks.

The networking services support a number of standard networking daemons such as DNS and DHCP.

Today, a cellular phone can use GPRS to access packet-data networks, including the Internet. There are two elements of the GPRS that are relevant to using GPRS to access these packet-data networks: the serving GPRS support node (SGSN) and the general GPRS support node (GGSN). The GGSN helps the cellular phone to access the various packet-data networks, as it acts as a gateway between the GPRS network and external networks. Figure 10 shows a GPRS connection to the Internet.

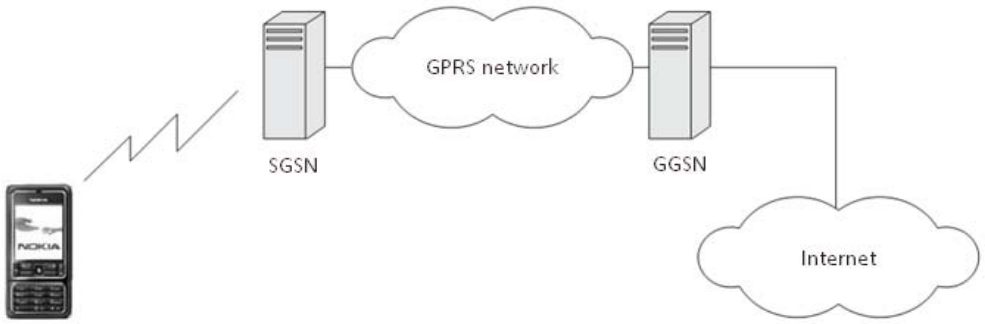

Figure 10: GPRS Connection to the Internet

The concept of an Access Point (AP) is one of the most important elements of the network configuration information. This concept is not only used for GPRS connections, but also other packet-data networks, such as Wi-Fi. This configuration information consists of the settings defining how to connect to a particular network. The information usually consists of an Access Point Name (APN), user name, password, and bearer type. The APN is sent to a SGSN by the cellular phone. The SGSN discovers a GGSN responsible for this APN. The GGSN is responsible for the interworking between the GPRS network and external packet switched networks, such as the Internet. There are a lot of GGSNs in the world. A mobile operator may provide several GGSNs which connect to different kinds of networks; including external network such as the Internet, the operator's internal network, and some specific service networks such as the MMS network. The (optional) user name and password are used for authentication.

Different Access Points can be used on conjunction with different services. The list of these APs is stored in the communication database of the cellular phone. It is important to choose an Access Point appropriate to the type of network service(s) the application needs. Additionally, multi-homing may be necessary. Sometimes, operators provide different services over different interfaces. For instance, MMS service is only available over a connection to the operator's MMS network. Thus the Symbian OS requires that the application declare which interface it needs to use, then it binds a socket to that connection using the RConnection API. The following steps are necessary to use an IP network connection:

1. Make a connection to the Socket Server using the RSocketServ API.
2. Use the Socket Server handler to open an RConnection instance (a connection handler).
3. Use the Connection handler to connect to the IP network. The application is allowed to specify a connection preference. Additionally, the application could define the specific AP by specifying the APN, user name, password, bearer type, and so on. Alternatively, an application could query the communication database to fetch a suitable AP. However, the filter criterion for APs needs to be defined by application developer. These criteria could be

any information about an AP. If the application does not specify the AP, then the user will be prompted to select the AP they want to use.

4. Do a data transaction using TCP, UDP, or using the HTTP framework over the connection.
5. After the application is finished using the network, then the connection handler should be closed.

In this thesis project, network services and the HTTP framework (see Section 6.1) are used together.

7.2.3 Messaging Services

The Messaging services give the user access to the messages which are owned by the Message Server. SMS (Short Message Service), MMS (Multimedia Message Service), and email are three typical types of messages in mobile phones. The Message Server supports storing and retrieving messages using the message store, notifying multiple applications of changes to the message store, and allowing multiple applications to receive, create, and send messages.

To use these message services, the application needs to open a session to the message server. The session is encapsulated by the CMsvSession class. The classes owning a CMsvSession usually implements the MMsvSessionObserver interface. The method HandleSessionEventL( ) in the interface will be called by the message server.

The message store uses a tree structure to store messages. Every node in the tree structure is represented by an entry. Each entry can be one of four different types: folder entry, message entry, service entry, and attachment entry. [22]

CMsvEntry is the key for the Message Services. It can be regarded as a pointer to the entry in the message store. So the application can do change, copy, move, and delete operations on the CMsvEntry. For instance, the method RSocketServ::GetEntryL(KMsvGlobalInBoxIndexEntryId) can be called to get an instance of CMsvEntry. KMsvGlobalInBoxIndexEntryId is the id of the inbox folder in the message store. After getting an instance of CMsvEntry, the application can call CMsvEntry::ChildDataL( ) to get the TMsvEntry instances which represent the message entries. At this point, the application can perform operations such as reading, modifying, and deleting messages.

7.2.3.1 Sending an SMS

The following steps are needed to send an SMS message:

1. Connect to the Socket Server (RSocketServ), and open a socket (RSocket) for SMS.
2. Set TSmsAddr::ESmsAddrSendOnly (TSmsAddr is the class represents the SMS address for the socket) as the SMS address family for the socket.
3. Insert the SMS body into CSmsBuffer (the class for storing the SMS text), and associate CSmsMessage (the class represents a complete SMS message) with the CSmsBuffer.
4. Set properties for the CSmsMessage, such as SMSC (Short Message Service Center) address.
5. Use RSmsSocketWriteStream class to send the SMS through the socket.
6. Call RSocket::Ioctl( ) method to apply an asynchronous I/O control operation. So that when the SMS is sent, the RunL( ) method will be called. (RunL( ) is the callback method for Active Objects, see section 3.4).

7.2.3.2 Receiving an SMS

The Symbian OS organizes the messages into a tree structure which included at least three different messages to different folders: Inbox, Sent, and Drafts. Thus, an incoming SMS message will be put in the Inbox folder, while an outgoing SMS message will be places in the Sent folder.

An application can register to be notified when entries are added, changed, or deleted. For example, we consider the case of receiving an SMS message: a messaging observer can be waiting for a new SMS message to appear in the inbox. When a new SMS arrives in the Inbox, a method called HandleSessionEventL( ) would be called. The application can now perform some operation such as check if the new incoming SMS message is specifically for this application.

However, this approach has a main drawback: **all observers** in the system are notified about the SMS message; adding, changing, or deleting. While this means that the user may be alerted to the arrival of a new SMS, for a message based application would be better to receive and process the incoming SMS message quietly without any disturbing the user. As a result, the application needs to intercept the incoming SMS message before it arrives at the message store. Fortunately, this is supported by the SMS Socket API. There are three phases for receiving an incoming SMS message: (1) waiting for the application specific SMS message, (2) intercepting the specific SMS message and processing it, and (3) acknowledging the received SMS message.

In the first phase, the application connects to the socket server (RSocketServ), then opens a socket (RSocket) for SMS. Similar to sending an SMS, the RSocket::Ioctl( ) method is called request an asynchronous I/O control operation. Thus when the new message arrives, the RunL( ) method will be called. A specific prefix can be bound with the SMS socket. For instance, assume there is an application waiting for receive a soccer match score via SMS, this match score SMS starts with string “MATCHSCORE”. TSmsAddr::SetTextMatch(_L(“MATCHSCORE”)) can be used to set the SMS prefix that will be applied to arriving messages. Based upon this filter the SMS messages starting with “MATCHSCORE” will be intercepted before reaching the Inbox.

In the second phase, once the SMS message starting with the specific prefix is intercepted by the application, the application can process the SMS message. This might include computing an abstract of the message and updating the user’s screen.

In the third phase, the application must acknowledge the incoming SMS message. This acknowledgement is needed so that the SMS stack knows that the SMS has been dealt with successfully, hence it can be deleted from the reassembly store. Otherwise, the SMS stack will attempt to deliver the message again.

8. Java Platform, Enterprise Edition

Java Platform, Enterprise Edition (Java EE) builds on the solid foundation of Java Platform, Standard Edition (Java SE) [29] and is the industry standard for implementing enterprise-class Service-Oriented Architecture (SOA) [30] and next-generation web applications. [31] It consists of a set of services, APIs, and protocols that provide the functionality for developing Web-based applications.

The following are the key features and services of J2EE which are relevant to this project:

- It supports standard HTML, Java applets, and Java applications.
- It uses Java Server Pages (JSP) and Servlet code to create HTML or other data for the web-client.
- It uses Java Database Connectivity (JDBC). It uses JDBC as an interface to communicate with a database which could be MySQL, Oracle, Access, or other similar database.
- It supports using JBoss as the application server. This makes developing web applications easier.

JBoss is an open-source Java EE based application server. It standardizes the application development architecture by defining several component models. These components include Enterprise JavaBeans (EJBs) [32], Java Server Pages (JSP), Java Servlets, and so on.

Without JBoss, the application always starts by executing a main method. Additionally, if the application needs to access various services, the developer has to write the code to handle the communication with the corresponding server(s).

However, with JBoss, the developer uses models to write the code, package it into a standard archive format, and deploy it to JBoss. Instead of accessing the services by writing code, the developers only need to provide the metadata in the form of Extensible Markup Language (XML), then JBoss uses the service managers and frameworks to access the services for the application.

8.1 Security in JBoss

JBoss uses the Java Authentication and Authorization Service (JAAS) API for user authentication and authorization. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework and compatibly extends the Java 2 Platform's access control architecture to support user-based authorization. [33]

There are three key concepts for the security in JBoss: ‘username’, ‘role’, and ‘security domain’. The ‘username’ and the ‘role’ are a pair used for representing a user. And the ‘security domain’ is an abstract concept which defines the authentication, authorization, and mapping modules. It is specified in the file ‘jboss-web.xml’ as following:

```
<security-domain>java:/jaas/exampledomain</security-domain>
```

In the above declaration, the ‘exampledomain’ is this security domain’s name. Once the security domain is defined, all the resources put in the domain will be protected. If any resource is requested without authentication, the user will be redirected to the authentication page. A resource is only accessible when

the user logs in with the relevant role. If the user is not allowed to access the resource, then the HTTP status code 403 will be returned in the response to the request.

Database based authentication can be used in this project. There is a file called ‘login-config.xml’ to define the configuration for the database based authentication in JBoss. The following is an example of an authentication policy:

```
<application-policy name=" exampledomain">
    <authentication>
      <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule"
        flag="required">
        <module-option name="dsJndiName">java:/OptiCallMySqlDS</module-option>
        <module-option name="principalsQuery">
          select user_password from users where user_name=?
        </module-option>
        <module-option name="rolesQuery">
          select user_role, 'Roles' from users where user_name=?
        </module-option>
      </login-module>
    </authentication>
</application-policy>
```

In the above example, the name for the ‘application-policy’ should be the same as the name of the ‘security domain’ defined in ‘jboxx-web.xml’. The database login module of JAAS is specified as the login module in the example. The value of the module option ‘dsJndiName’ is the data source name for the database, this will be discussed in the next section. The values for the module options ‘principalsQuery’ and ‘rolesQuery’ are the SQL query for verifying the user’s ‘username’ and ‘role’ respectively. It is worth mentioning, these are not standard SQL query statements , but rather are the variant of query statements used for JDBC.

The roles are defined in the file ‘web.xml’. An example of a role definition is:

```
<security-role>
    <role-name>administrator</role-name>
</security-role>
```

The access policy, called ‘security constraint’, is also stored in the file ‘web.xml’. Below is an example of a ‘security constraint’.

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>example area</web-resource-name>
      <url-pattern>/usermanager</url-pattern>
      <url-pattern>/groupmanage.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>manager</role-name>
      <role-name>administrator</role-name>
```

```
        </auth-constraint>
    </security-constraint>
```

In the above example, the value of the ‘web-resource-name’ is the name of the protected web resource in this constraint policy. The value(s) of the ‘url-pattern’ are the protected web resource’s URLs. In addition, the value(s) of the ‘role-name’ is the role with which a user can access this web resource. The above policy means that only the users with a role of ‘manager’ or ‘administrator’ can access the web pages ‘./usermanager’ and ‘./groupmanager.jsp’.

Additionally, the configuration for the login method needs to be defined in the file ‘web.xml’ for authentication. The following is an example of how to configure a login method:

```
    <login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
          <form-login-page>/login/login-form.jsp</form-login-page>
          <form-error-page>/login/login-error.jsp</form-error-page>
        </form-login-config>
    </login-config>
```

The value of the ‘auth-method’ can be ‘BASIC’ (browser-specific popup window), ‘FORM’ (for based authentication using a login page specified by <form-login-page> and the user is redirected to an error page specified by <form-error-page> on unsuccessful logins), ‘DIGEST’ (MD5 digest based authentication), or ‘CLIENT_CERT’ (Client-side SSL certificate based authentication).

8.2 Java DataBase Connectivity

Java Database Connectivity is a standard library for accessing databases. It can establish a connection to database, and allows an application to initiate a query or update statements to the database. Additionally, the statements can be parameterized so that duplicate statements can be avoided. The result will be stored in a table as metadata. Figure 11 shows the architecture of JDBC. [34]

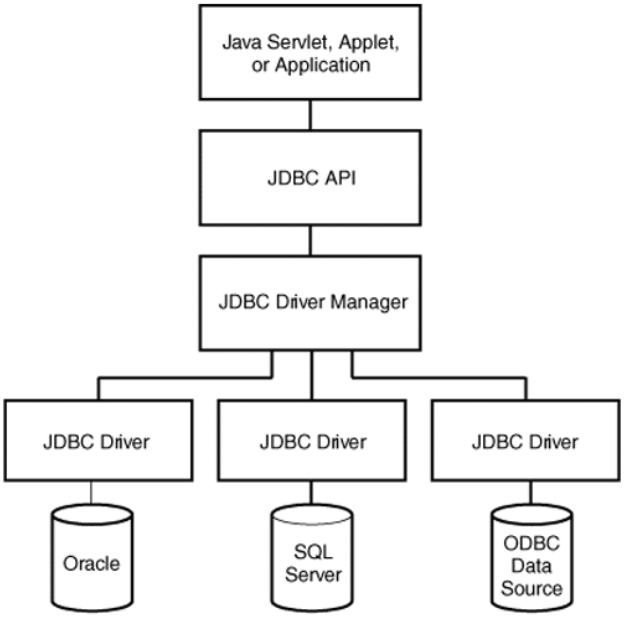

Figure 11: JDBC Architecture [34]

JDBC consists of two key parts: the JDBC API and a JDBC Driver Manager. [35]

JDBC API it provides programmatic access to the relational data from Java. Using the JDBC AIP, the applications can access the database, execute the SQL statements, and retrieve results and so on.

JDBC Driver Manager it defines the objects which can connect the application to a JDBC driver. So the application can communicate with vendor-specific driver which perform the real communication with the database. The driver translates the JDBC statements to the vendor-specific format so that the database server does not need to change anything.

Additionally, there is an alternative way to connect an application to the JDBC driver: DataSource. [36] A DataSource object represents a data source which can be a database or a file. Compared to the JDBC Driver Manager, a data object works with a Java Naming and Directory Interface (JNDI) naming service and is created, deployed, and managed separately from the application. Thus a DataSource can be shared by a number of different applications, thus the configuration has to be performed for JBoss rather than for a specific application. Normally, a driver vendor provides a class implementing the DataSource interface as part of JDBC. The following is an example of a specification for a DataSource:

```
<datasources>
  <local-tx-datasource>
    <jndi-name>OptiCallMySqlDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/exampledatabasename</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>root</user-name>
    <password>XXX</password>
    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSort
er</exception-sorter-class-name>
    <metadata>
      <type-mapping>mySQL</type-mapping>
```

```
        </metadata>
      </local-tx-datasource>
    </datasources>
```

In the DataSource configuration, the value of the ‘jndi-name’ is the name for this data source. The application needs to identify the DataSource by a name, in the declaration in the authentication configuration in the file ‘login-config.xml. A MySQL server was used in the above example, so the JDBC MySQL driver and metadata type mapping need to be set to MySQL.

9. AT Commands and SMSLib

9.1 AT Command

AT commands are also known as the Hayes command set [37]. These commands are sent from data terminal equipment (such as PC) to a modem (a type of data communications equipment) while the modem is in a command state. It is a de facto standard language for controlling modems. Most modems implement a standard set of AT commands. Using AT commands the data terminal equipment can issue commands for dialing, hanging up, changing parameters of the connection, etc.

For GSM modems, such as a GSM gateway and mobile phones, AT commands can be used to dial a call, send SMS/MMS messages, and change the device's PIN code, and so on. Below is the example for showing how to send an SMS message using AT commands from a PC:

AT
OK
AT+CMGF=1
OK
AT+CMGS="1234567"<*CR*>SMS example.<*Ctrl+z*>
OK

The ‘bold’ characters are the response string from the GSM modem. The first AT command tests and synchronizes the connection between the PC and the modem. The command "AT" stands for "Attention". The second AT command is to set the modem to SMS text mode. The third AT command is sends the SMS message by defining the recipient’s phone number and the SMS message’s content. <Ctrl+z> ends the message body for this command.

9.2 SMSLib

SMSLib is a Java library which enables users to send and receive SMS messages via a GMS modem or GSM phone. [38] It uses AT commands to communicate with the modem or phone (acting like a modem), so it is compatible with most GSM modems and GSM phones. Additionally, it supports several bulk SMS operators for outbound messaging. In this project, the provisioning server uses the SMSLib API to send the provisioning SMS message.

The following features of SMSLib were relevant to selecting it for use in this project:

- Supports communication with most GSM modems and phones via serial port interfaces.
- Supports simple and multipart inbound and outbound text messages.
- Supports sending SMS in PDU and TEXT mode.
- Supports 7 bit, 8 bit, and Unicode message encoding.
- Supports Outbound WAP PUSH SI messages.
- Supports Status (Delivery) Report messages.
- Supports fetching basic GSM information.
- Supports encrypting the SMS messages and other information.

There are six steps to required send a normal SMS message using SMSLib. These steps are

1. Construct a SerialModemGateway object which represents a GSM modem or phone connected via a serial port. The object id, the serial port number, the baudrate, and the SIM card PIN need to be set. Note that this is necessary even in the case of GSM phone, where there is no serial port being used - but the logical connection used by the software is as if it were using a serial port.
2. Construct a Service object which is the object of the main library class and add the SerialModemGateway object initiated in the previous step by calling the method Service.addGateway( ).
3. Start the Service by invoking the method Service.startService( ).
4. Construct an OutboundMessage object which represents an outbound SMS message. The recipient's phone number and the SMS content need to be set.
5. Send the SMS message by invoking the method Service.sendMessage( ).
6. Stop the service by calling the method Service.stopService( ).

However, SMSLib has some drawbacks:

1. Once started the connection cannot be disconnected before the service is stopped. Otherwise, the application will keep printing ‘WaitCommEvent: Error 31’ in console. The only way to stop it is restarting the application. This is the hardware problem. In this project, we keep the connection as stable as possible.
2. After the service is started, if the GSM modem or the GSM phone receives an incoming call, the application will keep throwing exception to the console. In some cases, the PC will even crash. This is a fatal problem when SMSLib is used in the provisioning system. Fortunately, this problem can be gone around in this project (see Section 11.4.2).

10. OptiCaller

The OptiCaller application is the client side of the DoD solution. Opticaller is designed as a cost saving calling application. It has been developed for Symbian based mobile phones. It supplies the user with alternative ways for cost effective calling, specifically via Call Back and Call Through. Additionally, as mentioned before, unlike the Web interface client and the old Java client, the OptiCaller supports the concept of a Mobile Extension as well as avoiding the user having to change their behavior.

10.1 Development Profile

OptiCaller is an application designed for running on a Symbian S60 mobile phone. Table 6 shows the OptiCaller development profile.

Table 6: OptiCaller Development Profile

UI platform	S60
SDK	S60 3rd Edition, Future Pack 1 (Symbian 9.2)
IDE	Carbide.c++
Debug Tool	CodeScanner, Error Reader
Test Module	Nokia E61, Nokia E51, Nokia E66, Nokia E71, Nokia N95

As introduced in Section 5.1, there are two main UI platforms for the Symiban OS: S60 and UIQ. OptiCaller is designed for the S60 platform, i.e., Nokia mobile phones and some mobile phone (modules) produced by Panasonic, Samsung, and others.

The software development kit (SDK) that was used is Symbian S60 3rd Edition, Future Pack 1. The operating System supporting this SDK is Symbian 9.2. However, in testing the OptiCaller application also works well on mobile phones based on Symbian 9.1 which utilized the Symbian S60 3rd Edition.

Carbide.c++ is a family of interactive development environments (IDEs) for the creation of C++ and C applications for Symbian OS devices. [39] This IDE is based on the Eclipse IDE and the C/C++ development tools from the Eclipse C/C++ Development Toolkit (CDT) Project. [41] It is based on Eclipse IDE Version 3 with the plug-ins which make Eclipse IDE understand how to handle Symbian C++ source files and build Symbian projects. [42]. The plug-ins can be classified into the CDT (which enables Carbide.c++ to manage and build C/C++ projects), Symbian Plug-ins (which make Carbide.c++ capable of supporting Symbian OS SDKs), and Nokia Plug-ins (which provide further support for Symbian OS SDKs). Figure 12 shows the interface of Carbide.c++ IDE. As should be clear from the figure, Carbide.c++ IDE is very similar to the Eclipse IDE.

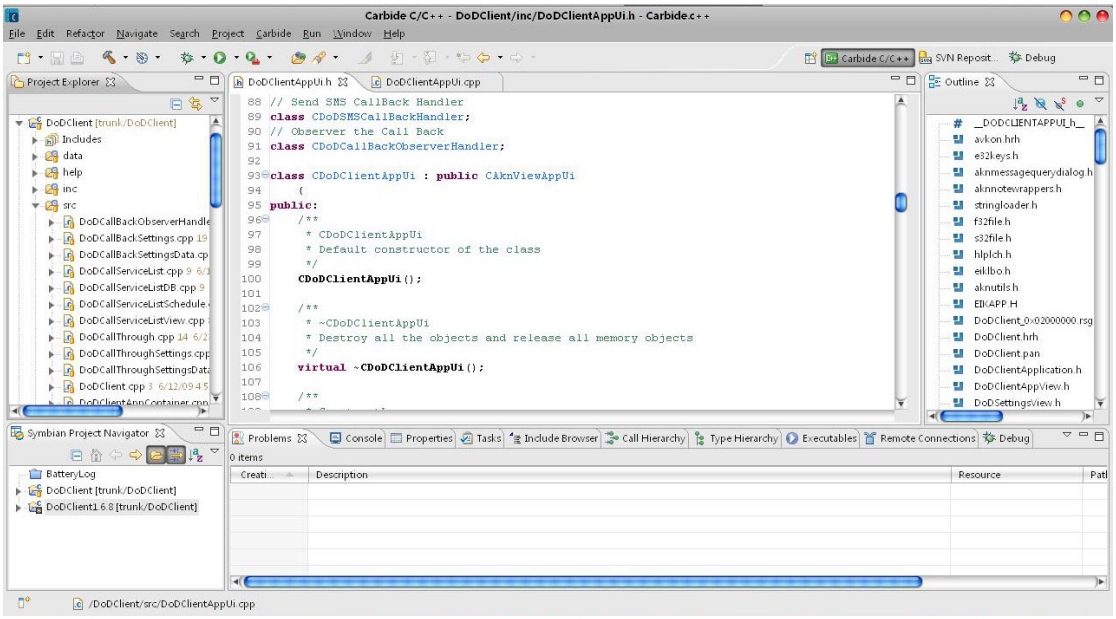

Figure 12: Carbide.c++ Interface

CodeScanner is a static source code analysis tool for Symbian OS programs written in Symbian C++. It delves deep into the Symbian C++ code by examining it line by line for the sources of defects. It is able to identify C++ coding convention deviations, incorrect descriptor usage, cleanup stack errors, and many other subtle problems that would be hard to track down. It is very useful for avoiding potential memory leaks. As described in Section 3.3.1, calling a leaving method in a non-leaving method could cause a memory leak, however, by using CodeScanner, this potential problem can be easily detected and avoided.

It is hard to debug a Symbian application. The normal way to debug an application is using an emulator running on a desktop or laptop computer. Unfortunately, the emulator does not support advanced APIs, such as the Telephony API unfortunately. Although Carbide.c++ provides on-device debugging, it is too slow for a mobile device to debug a complicated application. Therefore, the only way to develop such applications is to build the project, download the application to the mobile phone, then run the application to see if it works correctly. However, if the application crashes without any notification of something being wrong, then the developer must example a file called ‘ErrRd’ located in the ‘c:\resource\’ directory of the mobile phone. When an application crashes, the mobile phone should display extended error information instead of terminating the application without any notification. The useful information includes the exact reason and the error code. For released applications this could be distracting for the end-user, but is very helpful for the developer. However, it is no longer possible to create the ‘ErrRd’ file in the resource-directory manually - as this directory is not accessible for end-users. Fortunately, Error Reader is a small application which will create the ‘ErrRd’ file on the device in the appropriate directory. This application is available from ‘http://symbianresources.com/cgi-bin/schlabo/dl.pl?ErrRd’. The application can also be used to display error messages that have been written to this file. Figure 13 shows an example error notification as displayed by the Error Reader. Table 7 indicates the SDKs supported by the tested mobile phone modules. [43]

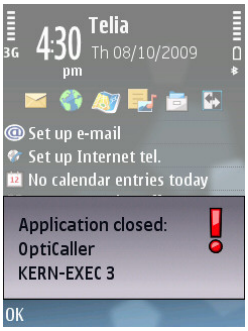

Figure 13: Error Reader Notification

Table 7: Tested Modules

Module	SDK version
Nokia E61	Symbian S60 3rd Edition (Symbian 9.1)
Nokia E51, E66, E71, Nokia N95	Symbian S60 3rd Edition, Future Pack 1(Symbian 9.2)

10.2 OptiCaller Overview

OptiCaller is designed to allow users to make a low cost call instead of a normal call without requiring that the user change their calling behavior. Figure 14 shows the interface of to OptiCaller when running on a Nokia E66 phone.

Figure 14: OptiCaller Interface

When the user makes a call with the OptiCaller running on the background, OptiCaller will intercept the dialed number, i.e., the callee's number, then the application will make a call to the cost-saving call service provider, such as DoD server. (This operation is done by using Telephony Services, as described earlier in Section 7.2.1) Thus from the caller's point of view only three steps to make a call using OptiCaller:

1. Dial the callee's number
2. Select an alternative way for making this call
3. Wait for the call to be established

The only extra step when making an alternative type of call is selecting the calling method that the user would like to use. In some cases, this step can be skipped by setting the configuration of the OptiCaller application.

There are three ways to make a call using OptiCaller: Direct Call, Call Back, and Call Through. In Direct Call, the call will be made as a normal call. The other two ways are cost-saving calling methods which will be discussed in the following sections.

The Figure 15 shows the user selecting the configuration for the calling method. In addition, to these three methods, there is one more option called ‘Always Ask’. If ‘Always Ask’ is set in the configuration of the application, then the user will be prompted to choosing the calling method for each call. Figure 16 shows the user being prompted to select the calling method for this call. If one of the other three options is selected, then the calling procedure will skip the extra step, so that the user makes a call as normally, but the call will set up using the selected alternative method. Note that some calls should always be made directly (i.e., 112, 911, etc.). For legal and safety reasons these calls will always be made using the direct method irrespective of the current setting of the Opticaller application. The list of numbers that receive this special treatment are built-into the application..

Figure 15: Call Method Setting Interface

Figure 16: Selection Call Method during Call

For reminding the user that the OptiCaller application is running in the background and which calling method has been selected, OptiCaller shows an indicator at the middle bottom of the screen. Figure 17 shows the mobile phone’s desktop when OptiCall is running in the background, here ‘AA’ stands for ‘Always Ask’, ‘DC’ stands for ‘Direct Call’, ‘CB’ stands for ‘Call Back’, and ‘CT’ stands for ‘Call Through’.

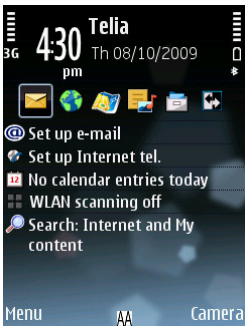

Figure 17: OptiCaller on the background

10.3 Call Back Implementation

10.3.1 Introduction to Call Back

In telecommunications, a call-back occurs when the originator of a call is called back immediately by a second call in response to their call. Normally, the call-back service provider has a unique number which must be first dialed in order to trigger a return call. A caller first dials a call to the service provider's number. On detecting an incoming call, the call will be rejected on the service provider's side. However, the incoming call provided enough information for the service provider to make a call to the caller. On receiving this call back, the caller should answer the call and provide the callee's number to the service provider's system. Given the callee's phone number the service provider makes a call to the callee and bridges the two calls (the call between the service provider and the caller and the call between the service provider and the callee). A common means for implementing this form of call-back is called DTMF (Dual-Tone Multi-Frequency) call-back because the callee's number is sent using DTMF signaling. However, the concept of call-back is not limited to DTMF call-back.

The DoD solution utilizes another communication channel to send the service provider the required number (i.e., the callee's phone number) by using the data link instead of DTMF over the voice link. Figure 18 illustrates the procedure used to make a call-back call using the DoD solution. Figure 19 shows the call-back procedure as implemented by OptiCaller. Assume that 'Call Back' has been selected as the calling method. The user's initial dialing of the callee is skipped in the figure.

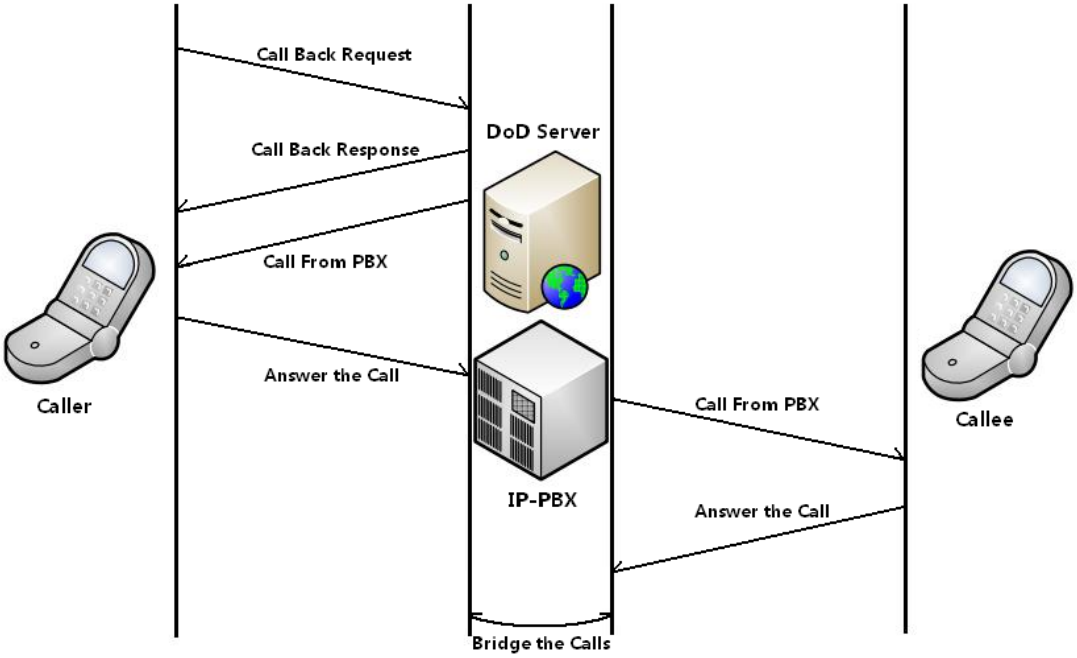

Figure 18: Call Back Procedure

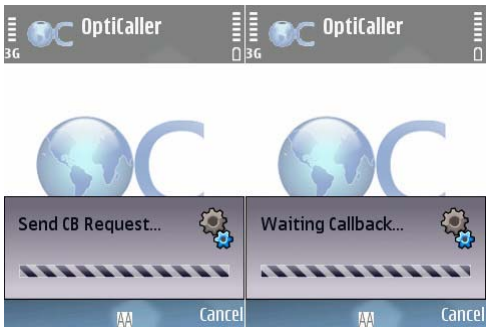

Figure 19: Call Back Procedure on the OptiCaller

In summary, the following steps are needed to make a call-back call:

1. OptiCaller intercepts the outgoing call, and stores the callee's number.
2. OptiCaller sends a request to the call-back server (the DoD server). This request could include authentication information (such as a username and password in the DoD solution), the call-back number to which the return call should be make, and the callee's number.
3. The user waits for the incoming call-back call.
4. OptiCaller answers the incoming return call automatically.

In DoD solution, users are allowed to initiate a call-back to a third-party's phone. This is designed to support some special (but useful) cases:

1. Assume the user both a roaming mobile phone and a local fixed phone. In this case the user can use their mobile phone to send a call-back request, but by providing the local fixed phone's number as the call-back number the user can use the local fixed phone to answer the call-back call.
2. The user can make a call for a third party, such as the colleague in the office, so that the colleague does not have to get the callee's number from the user and make the call by himself/herself. This second approach might be used by a call center manager to distribute

calls over a number of staff members, each of whom is to contact customers who have indicated that they want to be called (for example, this might be used with a web based customer support center to all the customer to click on a web page to get a call from the customer support center). In this second approach it is important that the information about this upcoming call also be provided to the colleague - so that they can properly handle the call (since they did not initiate it).

OptiCaller can use two different kinds of data-links to send the call-back request: HTTP/HTTPS and SMS (see Section 10.3.4). Both methods are supported by the DoD server. Thus in the DoD solution, users can select any suitable way to initiate a call-back - this might depend upon the user's circumstance for example, if GPRS is available, or which method is cheaper (with help from the provisioning system, see future work in Section 13.2.2). Additionally, OptiCaller is not the only application tailored for the DoD solution. Allowing alternative ways to initiate a call-back makes this solution compatible with other products. For instance, the users of 2N Netstar-Virtual PBX [44] can make a SMS call-back using the OptiCaller.

10.3.2 Call Back Configurations

Figure 20 shows the partial user interface for setting call-back configuration in OptiCaller. Table 8 shows the configurations for 'Call Back'. Some of the values are mandatory for specific configurations. The columns of 'HTTP' and 'SMS' indicate the mandatory configurations for the HTTP and SMS call-backs. One of 'Call Back 1' and 'Call Back 2' has to be set. And if the 'User Name' and 'Password' are mandatory depends on the solution requirement. If any necessary configuration is missed, the user will be prompt for setting the lacking configuration.

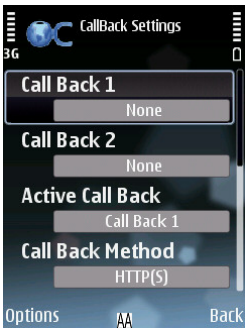

Figure 20: Call Back Setting User Interface

Table 8: Call Back Configurations

Configuration	Value	Description	HTTP	SMS
Call Back 1	(phone number)	The call-back number option 1	●	●
Call Back 2	(phone number)	The call-back number option 2		
Active Call Back	● Call Back 1 ● Call Back 2 (default: Call Back 1)	Either 'Call Back 1' or 'Call Back 2'. The selected one's number is set as the call-back number in the request.	●	●
Call Back Method	● HTTP/HTTPS ● SMS (default: HTTP/HTTPS)	Either 'HTTP/HTTPS' or 'SMS'. The selected one is set as the method to send the call-back request	●	●
Server IP Address	(IP address)	The call-back server IP address	●	
Server Port Number	(Port number)	The call-back server port number	●	
SMS Call Back Access Number	(GSM modem number)	The number of GSM modem used to receiving the SMS call-back request		●
SMS Format	(described in Section 10.3.4)	The format of SMS call-back request		●
User Name	(User name)	User name for authentication		
Password	(Password)	Password for authentication		

10.3.3 HTTP/HTTPS Call Back

OptiCaller uses Network Services (see Section 7.2.2) and the HTTP framework (see Section 6.1) to send an HTTP/HTTPS call-back request to the DoD server. As described in Section 7.2.2 Network Services, when the application uses the Connection handler to connect to the IP network, it first needs to connect to the Access Point (AP) provided by the provider. If there is no specific AP defined before it connects to the IP network, then the user will be prompted to choose an AP. Therefore, proper AP management is important to avoid annoying the user for each and every call. This consists of two parts: 1. testing the AP, and 2. storing an AP that has been used successfully. Figure 21 shows the procedure for making an HTTP/HTTPS Call Back call.

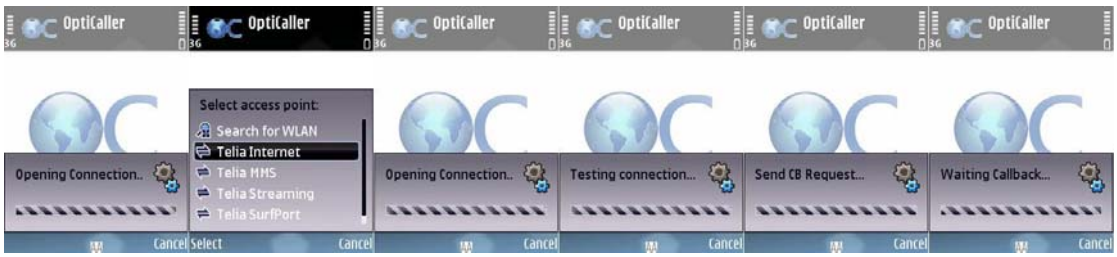

Figure 21: HTTP/HTTPS Call Back Call Procedure

As mentioned in Section 7.2.2 Network Services, the provider may supply several AP for users, including an AP for connecting to the Internet, an AP for streaming, an AP for MMS, and so on. Sometimes different services can share the same AP, for instance, the AP called 'Telia Surfport' provided by Swedish Operator 'Telia' [45] can be used both for connecting to the Internet and sending MMS. However, in most cases, the operators utilize different APs for different services. A user-friendly application should notify the user if the selected AP is suitable. To do this, OptiCaller provides an AP test function. When the user selects an AP for connecting to the Internet, OptiCaller tries to send a 'GET' request to the IP address of google website , as this is one of the most stable websites in the world, before sending a request to the DoD server. If the test gets successful response, this means that the selected AP is an Internet AP, thus the application should send its request to the DoD server via this AP. Otherwise, the user will be notified that the selected AP is not an Internet AP, then the user needs to select another AP.

There is a hidden setting which cannot be configured by human called 'Previous AP'. It is used for storing the AP ID in the cell phone's communication database. The ID of the most recently AP which was used for sending a request successfully is stored as the 'Previous AP'. If this AP is available, the user does not have to select an AP every time when they need to connect to the Internet. However, when the 'Previous AP' is unavailable, then the user must select another AP again. Although the setting is hidden, users can clear the 'Previous AP' setting by cancelling the request operation before the 'Waiting Callback' phase in Figure 21.

When OptiCaller tries to connect to the Internet, it first checks if the 'Previous AP' is available. If it is available, then OptiCaller uses this AP to connect to the Internet without requiring any human interaction. It is worth mentioning that before using the 'Previous AP', there is no testing of the AP, which reduces the cost a little, at the cost of sometimes being wrong about what AP should be selected. When necessary OptiCaller prompts the user to select an AP, then tests the selected AP before using it.

10.3.4 SMS Call Back

SMS is another way to send a call-back request using OptiCaller. OptiCaller uses the Message services (see Section 7.2.3) to sending an SMS Call Back request. In addition to the DoD solution, several other solutions support SMS based Call Back. The difference between SMS Call Back in these different solutions is the request format. In order to be compatible with multiple solutions, OptiCaller allows configuration for the 'SMS Format' message that is to be used.

This 'SMS Format' consists of characters which can be constant or variable. It includes special combinations of characters for inserting dynamic information. Table 9 shows these special combinations of characters.

When an SMS Call Back request needs to be formatted, OptiCaller will replace the special character combinations with the corresponding content. If any content is missed, the user will be prompt for the missing information. After formatting the request, OptiCaller sends the request to the number in the 'SMS Call Back Access Number' field.

For instance, an SMS Call Back request in the DoD solution consists of the action (which is 'smscallback'), the caller's number, the callee's number, the user name, and the password. The 'action'

field is used by the DoD server to determine what this SMS request is for. The 'SMS Format' is set as:

'action=smscallback&num1=/*N1*/&num2=/*N2*/&username=/*U*/&password=/*P*/'

When OptiCaller formats an SMS Call Back request, it replaces the special characters with the corresponding value and retains the other characters.

As mentioned before, an SMS Call Back request for the 2N Netstar-Virtual PBX consists of only the callee's number. For use with this PBX the 'SMS Format' is set as '/*N2*/'.

Table 9: Special Characters in SMS Format

Character Combinations	Description
/*N1*/	Need to be replaced by the number of the caller (also could be replaced by the number of the third-party)
/*N2*/	Need to be replace by the number of the callee
/*U*/	Need to be replaced by the user name for authentication
/*P*/	Need to be replaced by the password for authentication

10.4 Call Through Implementation

10.4.1 Call Through Introduction

Call Through is another service for making cheap international calls which is provided by many telecommunications companies. [46] Figure 22 illustrates the call through procedure. First, the caller needs to dial the access number to initiate the Call Through service (normally this is a local phone number for the caller). Once this connection is established, the caller dials the callee's number (including '00' and the country code). Given this information the service provider will make a call to the callee and bridge the two calls. For the caller, only the local calling fee (to the callee) is charged (although the caller may also have to pay for a call-through subscription).

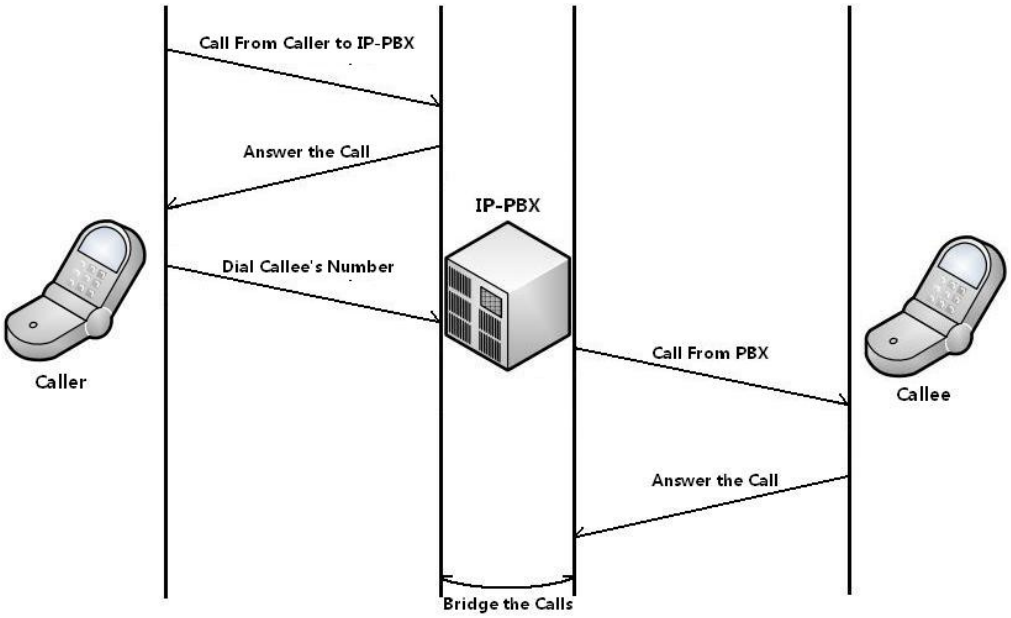

Figure 22: Call Through Procedure

However, this service needs more user interaction than making a normal call. OptiCaller enables the user to make a Call Through call just as a normal call. When the caller wants to make a Call Through call, he/she only needs to dial the callee's number and press the dial (i.e., 'green') button. OptiCaller will do the transactions between the caller and the service provider. The following steps are performed by OptiCaller:

1. OptiCaller intercepts the outgoing call, and stores the callee's number.
2. OptiCaller makes a call to the access number for the Call Through service, rather than directly calling the callee.
3. Once the call is established, OptiCaller sends the callee's number in DTMF tones to the service provider. (The DTMF tones may include the additional information such as a PIN code for authentication)

10.4.2 Call Through Configurations

Figure 23 shows the partial user interface for setting call-through configuration in OptiCaller. Table 10 shows the configurations for Call Through in OptiCaller. Except for the 'Access Number', the 'Dial Plan', and the 'Pause', other configuration fields are optional depend the value of the 'Dial Plan'.

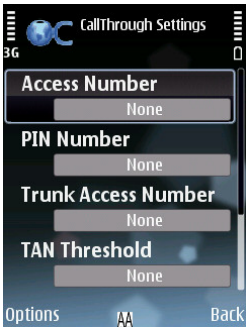

Figure 23: Call Through Setting User Interface

Table 10: Call Through Configurations

Configuration	Value	Description
Access Number	(phone number)	The access number for the Call Through service
PIN code	(PIN code)	The code for the authentication. It consists of the digits, '*', '#', and other characters can be sent by the DTMF tones such as 'p' (which means the pulse).
Trunk Access Number (TAN)	(digit)	(describe in Section 10.4.3)
TAN Threshold	(digit)	(describe in Section 10.4.3)
Dial Plan	● None ● PIN Only ● TAN Only ● PIN and then TAN (default: None)	(describe in Section 10.4.3)
Pause	● Null ● 1 Second ● 2 Seconds (default: Null)	The pause before each set of DTMF tones, such as the callee's number and the PIN code.

10.4.3 Call Through Dial Plan

OptiCaller was not designed for a specific Call Through service. It introduces a configuration called 'Dial Plan' to support different Call Through service providers (or Call Through PBXs). With different values of the 'Dial Plan', the DTMF tones are assembled in different ways. There are three configurations associated with the 'Dial Plan': the 'PIN code', the 'Trunk Access Number' (TAN), the 'TAN Threshold'.

'PIN code' The code for the authentication.

'Trunk Access Number' The trunk prefix is different from the trunk number to be dialed for a domestic call. The TAN determines which trunk the call should go through. For instance, if a person needs to make a call from an internal hotel phone to an external phone, he/she has to dial the trunk number (such as '9') to inform the PBX that the call is to an external number, this number is dialed before the actual number.

'TAN Threshold' Normally, the internal number has fewer digits than the external number. This threshold is used to determine if the call is to an internal number or an external number.

As mentioned before, there are four options for the 'Dial Plan': 'None', 'PIN Only', 'TAN Only', and 'PIN followed by TAN'. Each option leads different DTMF tone combinations being sent from the caller to the Call Through service provider. Table 11 shows the different operations of different dial plans.

Table 11: Dial Plan Operations

Dial Plan	Operations
Null	Send callee's number directly
PIN Only	1. Send PIN cods first 2. Send callee's number
TAN Only	1. Check the length of callee's number a) If it is shorter than TAN Threshold, it is an internal call b) Otherwise, it is an external call 2. Add Trunk Access Number ahead of external callee's number 3. Send the modified number
PIN followed by TAN	Combination of 'PIN Only' and 'TAN Only'

10.5 Mobile Extension Implementation

As described earlier, Mobile Extension (MEX) allows users to use their mobile phones in the same ways as they would use their fixed office phone. OptiCaller supports several MEX functions, making it better than the Web user interface and the old Java DoD client.

10.5.1 Call Through

Depending on the configuration of the PBX, Call Through can be regarded as a MEX function. By modifying the setting of the 'Dial Plan' to Call Through (see Section 10.4.3), users can use their mobile phones to make internal calls. In the DoD solution, each user's mobile phone will be associated with a fixed office phone (i.e., there is a mapping between a fixed office phone number and a given mobile phone number, and vice versa). When the user makes a call, the Caller ID will be set to be the number of the fixed office phone. The callee will see the internal number of the fixed office phone if the call is delivered as an internal call, or the external number of the fixed office phone if it is an external call). [6] Thus the call always seems like it is from a fixed office phone on the callee's side.

10.5.2 Three Lists – Presence, MEX, and Call Service

Today many PBXs support MEX functions such as setting presence status, login/logout groups, transferring calls, and so on. For instance, by setting presence status as 'Lunch Before 2 o'clock', the calls to the users before 2 o'clock will be blocked by the PBX (the PBX may notify the caller that the user is having lunch until 2 o'clock) so that the user will not be interrupted during their lunch. Alternatively, these calls could be redirected to another user who is responsible for answering phone calls during the other user's lunch break.

The PBX maintains table mapping different codes to different services. Using MEX functions is similar to making a Call Through call from the user's point of view. The user simply dials the access number of the MEX server and enters a code to indicate the function that they want to invoke.

However, some MEX services can only be used during an ongoing call, such as transferring calls. In these cases, the user does not need to call the MEX access number before sending a MEX code, but instead the solution takes advantage of the ability of the client to send data using a data link - even while the user might be using the voice link for a call. Instead OptiCaller maintains three lists for

different MEX services: ‘Presence List’, ‘MEX List’, and ‘Call Service List’. Each element of a list consists of two parts: the name (for user viewing) and the code (indicating the actual command). Figure 24 shows an example of a ‘Presence List’.

- ‘Presence List’: because there may be a lot of presence options, the OptiCaller uses a single list to store the presence options. The users can use the commands in this list to modify his/her presence status as seen by the PBX.
- ‘MEX List’: MEX services which require the user to call the MEX access number first are kept in this list; these include ‘Login Group’, ‘Logout Group’, ‘Do Not Disturb’, and so on.
- ‘Call Service List’: all the MEX services which are only used during an ongoing call are kept in this list; this set of commands includes ‘transferring calls’. In the case of the user issuing one of these commands, OptiCaller sends the service code to the PBX **without** dialing a call to the MEX access number.

Figure 24: Presence List

Each MEX service code consists of a static code and dynamic code. A static code indicates what MEX service the user wants to use. While the dynamic code provides additional information that the MEX service needs. OptiCaller utilizes a set of special characters to symbolize the dynamic information. Table 12 indicates the special characters of a MEX function code for OptiCaller. Before sending the MEX service code, OptiCaller will consider the selected item code, an if necessary solicit input from the user. Opticaller will replace the special characters with the user input.

For instance, assuming that the ‘Lunch’ presence code is *23* followed the expected ending time of the lunch. Additionally, we will assume that each of the commands ends with a ‘#’ symbol. If a user want to set their presence as ‘Lunch Until 2 o’clock’, without OptiCaller, he/she needs to dial the MEX access number, and press ‘*23*1400#’ manually after the call is established. However, using OptiCaller, the user simply places an item called ‘Lunch’ whose code is ‘*23*t#’ in the ‘Presence List’. When the user actually was to set their ‘Lunch’ presence status, he/she only needs to select the ‘Lunch’ item, and input the expected ending time (‘1400’ in this example). OptiCaller will send the request from the mobile phone to the PBX.

Table 12: Special Characters in Lists

Special Character	Description
‘t’ or ‘T’	This character should be replaced by the time the user input. The input format is HHmm. For instance, ‘1400’ means ’14:00’.
‘d’ or ‘D’	This character should be replaced by the date the user inputted. The input format is YYMMDD. For instance, ‘091011’ means ‘11st, September 2009’.
‘n’ or ‘N’	This character should be replaced by the phone number the user input. Only digits are accepted.

The ‘Call Service List’ is different from the other two lists. As mentioned before, the first reason is that to use the services in this list, OptiCaller sends the code to the PBX directly without dialing a call to the MEX access number. So there is a limitation for the services in the ‘Call Service List’ that the call has to go through the PBX which provides these services.

Additionally, one more special character is introduced into the ‘Call Service List’: ‘+’. In the case of the other two lists, OptiCaller collects all the dynamic information before sending DTMF tones, which may lead to problems for the ‘Call Service List’. For instance, the code for the service ‘Transferring Calls’ is ‘*25*n#’. Before sending the code, OptiCaller needs the user to input the transferring destination number to replace the character ‘n’. However, since the call is still active this means the other party of the call will hear the DTMF tones when the user enters the number. This is very annoyed, thus OptiCaller introduces the character ‘+’ to avoid this problem. When OptiCaller formats the service code, if it finds the character ‘+’, it will send the partial code which is ahead of the ‘+’ to the PBX, then continue formatting the rest code after the ‘+’. Therefore a service such as ‘Transfer Call’ can be modified into ‘*5#+*25*n#’ (assuming that ‘*5#’ is the code for a ‘Hold Calls’ service). In this case, OptiCaller will send ‘*5’ to the PBX to place the call between the PBX and the other party on hold first, then continue formatting the rest of the code so that when the user inputs the destination number, the other party will not hear the DTMF tones.

10.6 Implementing Provisioning

OptiCaller implements alternative ways for allowing the user to make low cost calls without changing the user’s calling behavior. Additionally, OptiCaller was designed to be compatible with several different PBX solutions. However, a result is that the configurations of OptiCaller are complicated. As OptiCaller should be an application used by end-users, its usage should be as simple as possible. So a provisioning system was developed to help users avoid dealing with complicated configurations.

10.6.1 Overview of Provisioning

The provisioning system consists of two parts: a provisioning server, a provisioning client (this is built into OptiCaller). Figure 25 illustrates the provisioning transactions between the provisioning server and OptiCaller. When there is configuration update for OptiCaller, the provisioning server will send a SMS message to the mobile phone on which OptiCaller runs. This SMS message starts with a specific sequence of characters and includes the IP address and port number of the provisioning server. After receiving such provisioning SMS message, OptiCaller will send its authentication information

(including the username and password) via an HTTPS POST request to the provisioning server. If the user is successfully authenticated, then the provisioning server will return the latest configuration information for this user's OptiCaller client in a response. This section focuses on the provisioning system from the client side. The provisioning functions in OptiCaller consist of three parts: configuration handling, waiting for a notification, and updating the application's configuration. Note that SMS is being used to send the provisioning message to the mobile phone, because many cellular operators block incoming data traffic, but do allow incoming SMS traffic. An alternative solution would be to have the clients check for updates at some pseudo random interval, but it will cause extra cost.

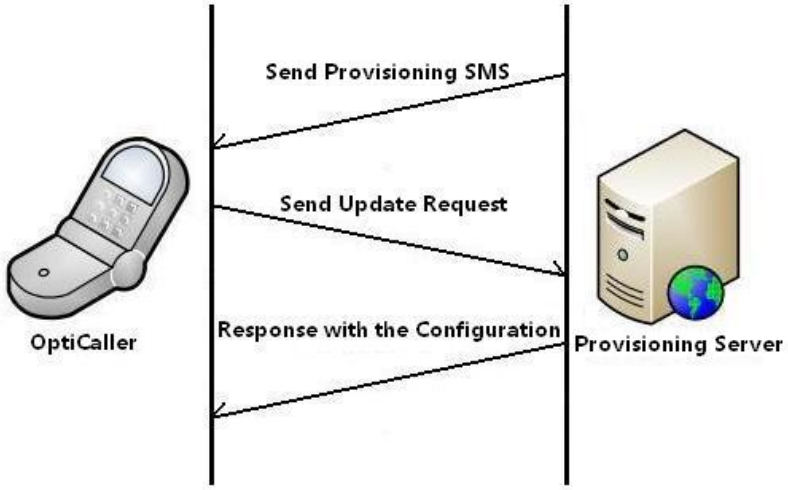

Figure 25: Provisioning Procedure

10.6.2 Provisioning Configurations

Table 13 shows the information needed for the provisioning procedure in OptiCaller. The first two are sent in the SMS message and the last two are either solicited from the user. They are stored in the local configuration information of the application.

Table 13: Provisioning Configurations

Configuration	Value	Description
Provisioning Server IP Address	(IP address)	The provisioning server IP address
Provisioning Server Port Number	(port number)	The provisioning server port number
Provisioning User Name	(user name)	User name for authentication
Provisioning Password	(password)	Password for authentication

10.6.3 Configuration Handling

The provisioning configuration of OptiCaller is stored in two files: ‘Configuraion.txt’ and ‘Private.dat’. The file ‘Configuration.txt’ stores the general configurations such as the current ‘Call Method’, the ‘Server IP Address’ for Call Back, the ‘Access Number’ for Call Through, and so on. This file is human accessible in the phone’s file manager. The file ‘Private.dat’ stores the user’s specific information, such as the username and password for Call Back, the username and password for the provisioning system. It also contains the background configuration such as the ‘Previous AP’ for the AP management (see Section 10.3.3). The content of the "Private.dat" file is specific for each user whichcannot be with other

users. It is in the OptiCaller application's private directory and is not human accessible. When OptiCaller is uninstalled, its private directory will also be deleted too -- thus private settings will be deleted, while the general information will continue to exist. Thus if and when OptiCaller is re-installed, the old general configuration will be available. Figure 26 shows part of the general configuration file on a Nokia E66 phone.

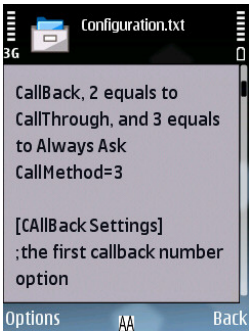

Figure 26: Partial Configuration on a Nokia E66

There are other two configuration files stored in the application's private directory (they are both inaccessible to the human user): 'DefaultConf.txt' and 'FormatConf.txt'. The file 'DefaultConf.txt' stores the application's default configuration. If the user selects the menu item 'Restore Default Set' the application's settings will be restored this default configuration. The file 'FormatConf.txt' is a template for the general configuration file 'Configuration.txt'. It is used by the application to rewrite the general configuration file.

OptiCaller loads its configurations from the files 'Configuration.txt' and 'Private.dat' when the application is launched. After successfully starting it writes the general configuration back to the file 'Configuraion.txt'. If there is an error in the 'Configuraion.txt' file the application will replace the contents of this file with the contents of the file 'FormatConf.txt', and write a new 'Configuration.txt' file. If any configuration value/setting is modified, the corresponding file is modified. OptiCaller will copy the 'DefaultConf.txt' to the directory, then rename it to 'Configuraion.txt' when the general configuration file is missing.

10.6.4 Waiting for an Update Notification

When OptiCaller is running, it monitors the incoming SMS message using the Messaging services (see Section 7.2.3). OptiCaller checks if the incoming SMS message is a provisioning SMS message by checking if the incoming SMS starts with the special sequence of characters '*OptiCallerProvision*'. If the provisioning SMS message is received, then OptiCaller will intercept the SMS message, read the provisioning server's IP address and port number from it, then save the provisioning server's information in the corresponding configuration field. Subsequently OptiCaller informs the user that there is an update available from the provisioning server, asking of the user wants to update their configuration now or at a later time. If the user chooses to update now, OptiCaller will start the updating procedure (see Section 10.6.1). Otherwise, OptiCaller will not start the procedure until the user actives the update operation by selecting the menu item 'Update Set'.

If that the provisioning SMS message is received when OptiCaller is not running on the user's mobile phone, then the contents of the provisioning SMS message includes information asking the user to leave the message for OptiCaller. When the OptiCaller is launched next time, it first scans the SMS

message in the ‘Inbox’ in order from the latest to the oldest. If it finds a provisioning SMS message, it extracts the provisioning server’s information from the SMS message, saves this information, and deletes the SMS message. It also deletes any duplicate provisioning SMS message in the ‘Inbox’. Only the information from latest provisioning SMS message is used. The user will be asked if they want to update now or later.

10.6.5 Updating the Application's Configuration

The updating process will start when the user chooses ‘Update Immediately’, on receiving an update notification, or when the user presses the menu item ‘Update Set’. As shown in Figure 25, in each of these cases OptiCaller sends an HTTPS request to the provisioning server and the server returns the latest configurations in response. The provisioning function shares the AP management mechanism (see Section 10.3.3) with the Call Back function, thus the user can update the application's configuration without any human interaction.

After receiving the latest configuration, OptiCaller stores the configuration into the file ‘DefaultConf.txt’, generates a new template file ‘FormatConf.txt’, and finally restores the general configurations based upon this new default configuration. The configuration correction check is maintained as the future work (see Section 13.2.1).

10.7 Other Functions

There is a list called ‘White List’ in OptiCaller. This list stores the numbers that the user does not want OptiCaller to make using an alternative calling method, these numbers include the emergency call and perhaps the cellular operator's voice mail box. All calls to the numbers on the ‘White List’ are always made directly.

Although the items in the ‘White List’ can be modified by the user OptiCaller is hard-coded to always make direct calls to emergency numbers, such as ‘911’ (Swedish Police) and ‘112’ (Swedish Ambulance) as a result these numbers are not shown in the ‘White List’ and these numbers cannot be modified or deleted by the user.

11. OptiCaller Provisioning System

As mentioned in the previous chapter, a lot of values need to be configured before OptiCaller can be used by typical end-users. Furthermore, OptiCaller needs a means to be deployed on the end-user's mobile phone. The OptiCaller Provisioning System was developed to handling both remote installation of the application and updating of the application's configuration.

11.1 OptiCaller Provisioning System Architecture

There are three roles in the OptiCaller Provisioning System: the 'administrator', the 'manager', and the 'end-user', each with different rights. Figure 27 illustrates the architecture of the OptiCaller Provisioning System. An important concept called 'group' in the OptiCaller Provisioning System is used to define a collection of end-users who share the same configurations. For instance, in a company, employees in the different departments may have different configurations when they use OptiCaller, while employees in the same department might share the same configuration. Table 14 explains these different roles in the OptiCaller Provisioning System.

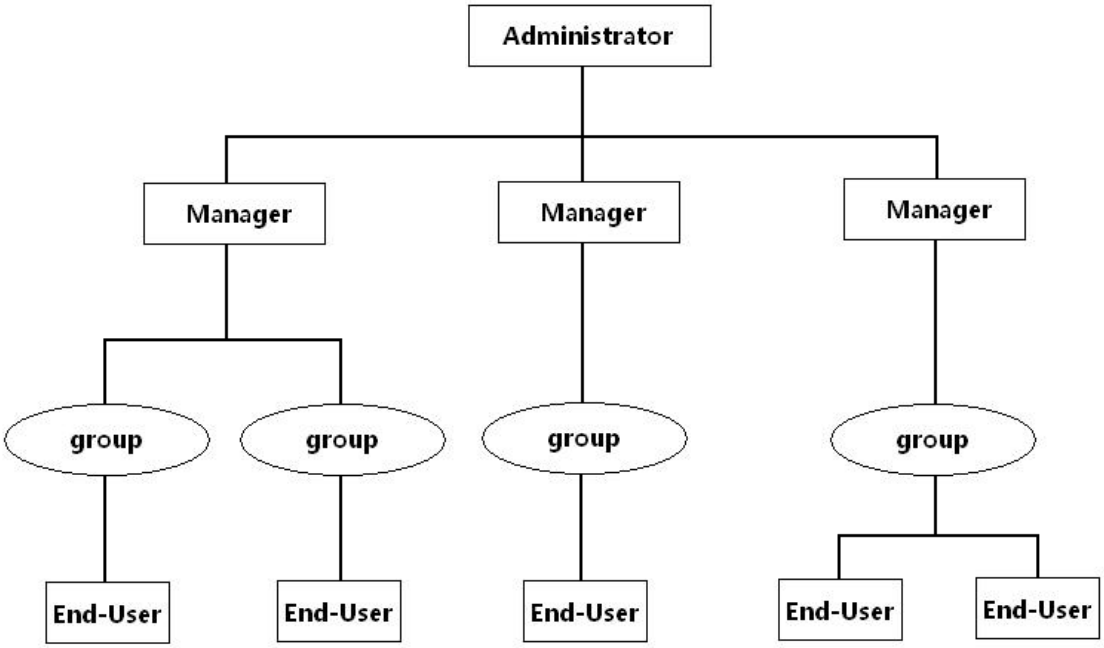

Figure 27: OptiCaller Provisioning System Architecture

Table 14: OptiCaller Provisioning System Roles

Role	Description	Rights
End-user	The direct users of OptiCaller	● Download and Install the OptiCaller ● Updating their OptiCaller configuration ● Manage their information, such as their password
Manager	The administrators in the companies who manage the company's end-users	● Manage a group, including creating a group, updating a group, and deleting a group ● Manage end-users, incl. adding a given end-user to a specific group and deleting the end-user from a group ● Remote control, notify the end-users where OptiCaller can be download from, or there is update for their copy of OptiCaller ● Manage information, such as the end-user's password
Administrator	The super administrator who manages all the users in the OptiCaller Provisioning System	● All the power of the 'manager' ● Manage the 'managers', including adding a given manager to the system and deleting a given manager from the system ● Manage the connections between the server and the GSM modem which is responsible for sending the provisioning SMS messages ● Manage the OptiCaller installation file

11.2 OptiCaller Provisioning Procedure

Before introducing the Provisioning Procedure, there are two kinds of provisioning SMS messages that need to be described: the 'Installation SMS' and the 'Update SMS'.

'Installation SMS' An installation SMS informs the end-user that OptiCaller is available to be downloaded and installed. This SMS message starts with the string '*OptiCallerInstallation*'. In addition, to the text for the end-user to read, there is a link in the SMS indicating where the end-user can download the OptiCaller installation file from. On a mobile phone that supports OTA installation, the end-user simply opens the link in the SMS message in their 'Inbox', then the download and installation process will start automatically.

'Update SMS' An update SMS informs the existing instance of OptiCaller (or it informs the end-user when OptiCaller is not running) that there is the configuration update available. As stated previous this message starts with the string '*OptiCallerProvision*'. In addition to the

text for the end-user to read, there is information about the OptiCaller Provisioning server, including its IP address and port number. When OptiCaller runs, it will start the updating process automatically. If OptiCaller is running, then the end-user will not even notice that the update SMS arrives- unless they are prompted to ask if the update should be done now or not. (See Section 10.6.1)

Figure 25 illustrates the Provisioning Procedure (except for the initial deployment of the application). There are five steps in each provisioning transaction. These steps are

1. The provisioning server sends the two provisioning SMS message to the end-users.
2. The end-users download and install OptiCaller (by opening the link in the 'Installation SMS' or in some other way installing the application).
3. OptiCaller sends an updating request, contains the authentication information, to the provisioning server when it finds an 'Update SMS' message.
4. The provisioning server verifies the end-user should be permitted to the access this configuration, then fetches the configuration for this end-user's group, and returns this configuration in its response.
5. The OptiCaller updates its configuration(s).

11.3 Provisioning Server Development Profile

The OptiCaller provisioning server is a web server which provides a platform for the OptiCaller Provisioning System. Table 15 indicates the profile of the OptiCaller provisioning server development.

Table 15: Provisioning Server Development software

Java Runtime Environment	jdk1.6.0_15
Web Container and Server Application	JBoss Application Server 5.0.1-GA
IDE	Eclipse IDE for Java EE Developers
Database Server	MySQL Server 5.1

JBoss is used as the application server, as it makes developing web applications much easier (see Chapter 8). The database server that we have selected is a MySQL server; because it is very easy to install and sufficiently powerful for this project. All the transactions between JBoss and MySQL are done by the JDBC (see Section 8.2). In this project, the provisioning server uses the Java Persistence API (JPA) [40] to handle the JDBC transactions.

The Eclipse IDE for Java EE Developers is the branch of the Eclipse IDE which wasdesigned for helping Java EE developers to develop the Java and Java Enterprise Edition applications. [41] It provides Java editing with incremental compilation, Java EE 5 support, a graphical HTML/JSP editor, database management tools, and support for most popular application servers (such as Tomcat and JBoss). Figure 28 shows the interface of the Eclipse IDE for Java EE Developers.

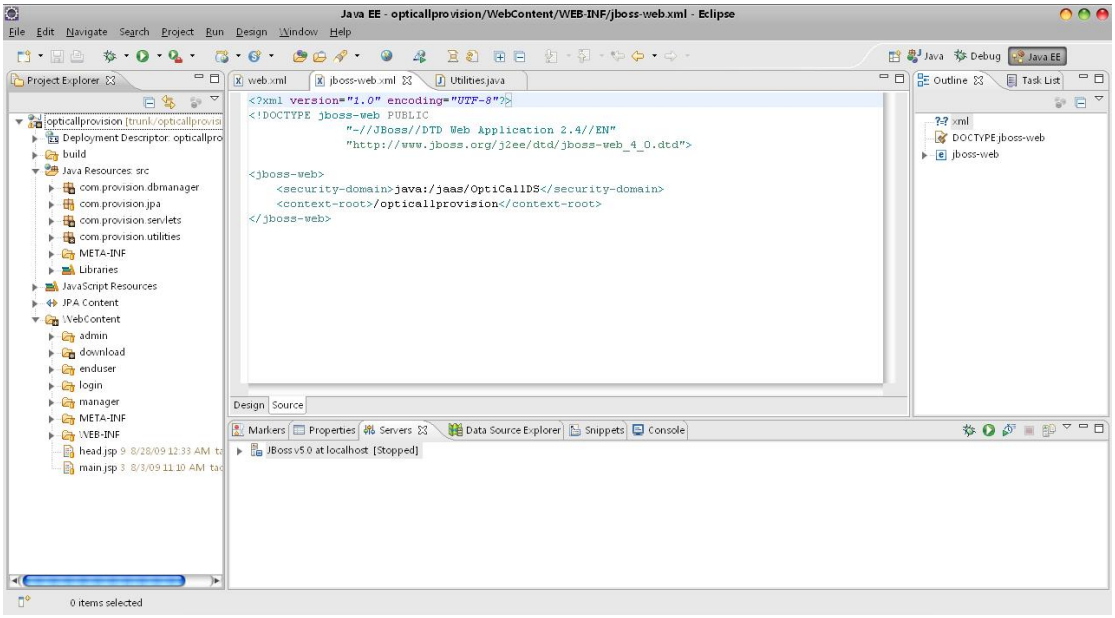

Figure 28: Eclipse IDE for Java EE Developers Interface

11.4 OptiCaller Provisioning Web Site

An OptiCaller provisioning web site is the platform for the OptiCaller Provisioning System activities. This section will explain the services provided by the OptiCaller provisioning web site for each of the different user roles. All the web user interface figures are included in Appendix B Figures 37 to 47.

11.4.1 End-User Services

As shown earlier in Table 14, there are three roles. The services for the end-user include downloading and installing the OptiCaller application, updating the OptiCaller's configuration, and updating other information. When downloading the OptiCaller installation file, there is no authentication of the user needed, thus anyone can download the installation file. However, before updating the configurations or modifying the end-users' information, the end-users must be successfully authenticated first (see Figure 37). The profile of each end-user is the username and password, and the mobile phone number (see Figure 38). The password and the mobile phone number can be modified by the end-users themselves.

11.4.2 Manager Services

Users with the role 'manager' can manage groups, including adding/deleting/updating a group profile and adding/deleting end-users to/from each group. All the resources for managers are placed in a security domain. Therefore, if the manager is not logged in, the user's requests will be redirected to a login page (as defined in the 'web.xml') (see Figure 39). If the manager is successfully authenticated, then the manager is allowed to use the resources, then he/she will be able to access the corresponding page (i.e., the page which provides the operations that the user is allowed to perform). Otherwise, the HTTP status code 403 will be returned in the response to the manager. More details are given in Section 8.1. Similar to the end-users, the managers can modify their password on the provisioning web page (see Figure 40).

The page 'Add Group' (see Figure 41) is used by managers to add a new group. To add a new group,

the manager simply enters the group's name, which must be unique in the groups managed by the manager. Additionally, the manager may set the default configuration for a new group. This configuration can be modified later. The configuration items are the same as the configuration of OptiCaller (see Sections 10.3, 10.4, and 10.5).

The page 'Update Group' (see Figure 42) is used by managers to update a group's profile, including modifying the group's name, the default configuration, and adding/updating/deleting end-users to/in/from the group, and deleting the group. When a manager selects an existing group from the list of all the groups managed by this manager, the browser will send a request to the server to retrieve the group's profile and display it.

After updating the group's profile, the manager can click the 'Deploy Group' button to reach the 'Deploy' page (see Figure 43). There are two lists shown on this page: the left one is a list of all the end-user related information in this group (including the username, the latest deployed time, and the user's phone number); while the right list contain the list of the end-users that need to be deployed at this time. The manager can move the end-users from the left list to the right list. After selecting the deployed end-users, the manager can send an 'Update SMS' message to these end-users simply by click the 'Deploy' button. Additionally, the manager can send an 'Installation SMS' message to the end-users at the same time by choosing the option 'Deploy OptiCaller'. The provisioning server uses the 'SMSLib' library to sending the relevant SMS (see Section 9.2). The second drawback of the 'SMSLib' library is a fatal bug in the provisioning system. In this project, we used call diverting/forwarding service [47] to overcome this problem. A 'Deploy Result' page (see Figure 44) will be presented after the SMS messages have been sent.

11.4.3 Administrator Services

The super administrator has all the abilities of any of the managers; hence he/she can do anything on behalf of a manager. The administrator has the ability to add/delete managers (see Figure 45).

There is a page for the administrator to manage the serial connection between the web server and the GSM modem which is used to send the provisioning SMS messages (see Figure 46). The configuration includes the com port number, the baudrate, and the SIM card PIN code.

The administrator also manages the OptiCaller installation file (see Figure 47). When a new OptiCaller release is ready for use by end-users, the administrator needs to upload the installation file to the provisioning server so that the end-users can download it. The OptiCaller provisioning server uses the 'Commons FileUpload API' [48] to provide this upload file service. The provisioning server uses some standardized information, such as the file extension and the file size, to verify the correct file has been successfully uploaded.

12. Evaluation

This project included the design and development of the OptiCaller application based on a Symbian mobile phone and the designing and implementation of the OptiCaller Provisioning System. To be released as a commercial product, in addition to testing that the functions being offered work, establishing the quality of the OptiCaller application and the OptiCaller Provisioning System are also important. This chapter illustrates the evaluation of the performance of the OptiCaller and the OptiCaller provisioning server according to a set of metrics that were chosen for each of these different subsystems. Additionally, drawbacks of current solution have been found which suggest future work for the project (see section 13.2).

12.1 OptiCaller Evaluation

As described in previous chapter, OptiCaller offers users several alternative ways to make an outgoing call. In this chapter we describe the evaluation of this application; both in terms of functional tests and the performance of OptiCaller with respect to two metrics call performance in terms of time to establish a call and the power consumption of running the application in the background. The first metric is important to the user as if the delay is too long when using one of the alternatives in comparison to a direct call, then the user is likely to avoid these alternatives and simply make direct calls. The second metric is important, since if the power consumption of running the application is too high, then the users will not run the application!

12.1.1 Test Equipments

The two tests are done on two Nokia S60 smart phones, models N95 and E61. The technical profiles of these two models are [43]:

- N95:
  - Operating Frequency: GSM850, GSM900, GSM1800, GSM1900, UMTS2100
  - Operating System: Symbian 9.2 S60 3rd Edition, Feature Pack 3.1
  - Battery Capacity: 970mAh
  - Talk time: up to 4 hours (2G), up to 2.5 hours (3G)
  - Stand-by time: up to 8 days (2G), up to 9 days (3G)
- E61:
  - Operating Frequency: GSM850, GSM900, GSM1800, GSM1900, UMTS2100
  - Operating System: Symbian 9.1 S60 3rd Edition
  - Battery Capacity: 1500mAh
  - Talk time: up to 5 hours (2G), or up to 3 hours (3G)
  - Stand-by time: up to 13 days (3G and 2G)

The network connection used for all tests was the ‘Telia SurPort’ Access Point. The tests were conducted in Kista, Sweden (as suburb of Stockholm). As a result the only operating frequencies that these devices could use were: GSM900 and GSM1800 for 2G service and UMTS2100 for 3G service. The phone was set to only use 3G service.

12.1.2 Call performance

There are three phases for users to make an outgoing call: 1. dial number, 2. wait for the call to be established, and 3. speak to the other party. The first phase was described in previous chapter. The critical requirement was that users can make a call without changing the dialing behavior -- as this is one of the most important advantages of OptiCaller. In the third phase, the voice quality is the most important aspect of the on-going call to the user, however this voice quality is out of the scope of the OptiCaller application as this depends on the performance of the mobile phone, the quality of the operator's service, and so on (note that the performance of the PBX must also be considered). As the time to dial a number is independent of whether the user is making a direct call or using OptiCaller, our measurements of call performance for OptiCaller focuses on the second phase, specifically the time to establish a call given the number of the callee. OptiCaller will be compared to direct dialing and the new Java client which supports call-through instead of the old Java client.

12.1.2.1 Test Scenario

Figure 29 illustrates the test scenario. It consists of two mobile phones: Mobile A and Mobile B. Mobile A is the caller which is installed the OptiCaller and Test Application 1-A, and Mobile B is the callee which is installed the Test Application 1-B. The test application 1-A records the time when Mobile A starts to make an outgoing call and the time when the outgoing call is answered by the other party. The test application 1-B answers all incoming call automatically. There are five tests:

- Mobile A calls Mobile B without running OptiCaller (i.e., a direct call).
- Mobile A calls Mobile B by Call Back with OptiCaller.
- Mobile A calls Mobile B by Call Through with OptiCaller
- Mobile A calls Mobile B by Call Back with the Java client.
- Mobile A calls Mobile B by Call Through with the Java client.

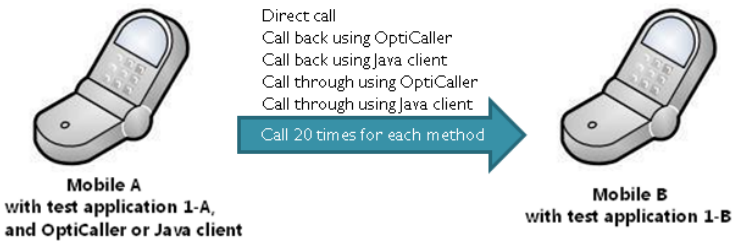

Figure 29: Call Performance Test Scenario

During the tests two to four, the configuration for the calling method in OptiCaller and the Java client are preset to the corresponding call method so that the time required for users to select the call method is zero, since this process is skipped in both cases.

Each test is repeated 20 times on both the Nokia N95 and Nokia E61.

12.1.2.2 Test Results

The goal of this test is to measure the delay due to using OptiCaller comparing to normally establishing a call with a mobile phone and compared to the time required to establish the call using the Java client. For each test, the call started time and the call answered time are one record. The difference between these two times is the delay for establishing call. Table 16 shows the results in terms of the average value of 20

repetitions for each test. According to Table 16, the variances are less than 1 second. This means the 20 repetitions are enough for the test. The complete test results are included in Table 20 in Appendix A.

Table 16: Call Establishment Test Results

Mobile Phone	Calling Method Average Time to Establish a Call (second)				
	Direct Call	Call Back		Call Through	
		OptiCaller	Java	OptiCaller	Java
Nokia N95	6.8±0.9	23.6±1.0	17.5±0.8	18.5±0.9	19.4±0.7
Nokia E61	6.7±0.5	16.5±0.8	11.0±1.0	18.6±0.5	20.5±0.8

12.1.2.3 Analysis of test results

In these tests the average direct call establish times for both the Nokia N95 and Nokia E61 are approximately 7.0 seconds, establishing the baseline for the other calling methods.

The Call Back call establishment time using the OptiCaller application on Nokia N95 is 23.6 seconds, while the Call Back call establishment time using the Java client on Nokia N95 is 17.5. Thus OptiCaller required around 6.0 seconds more than the Java client. The difference between the OptiCaller Call Back call establish time and the Java client Call Back call establish using the Nokia E61 are roughly 5.5 seconds. The major reason for this difference is that the OptiCaller application connects to the Access Point every time before sending the call-back request to the DoD server, then closes the connection after finishing the transaction; while the Java client only connects to the Access Point when the Call Back function is used for first time and does not close the connection until the application is closed. This was verified using another test application to measure the time taken to connect to the Access Point. According to this sub test, the Symbian application took approximately 5.0 seconds to connect to the Access Point (see Table 21 in Appendix A), hence the difference between the OptiCaller and the Java client results is reasonable. If rather than simply reporting the averages you had looked at the time for the first call versus the times for the subsequent calls you would have see that the first call took longer for the Java client, but ~5.0 seconds less for all the subsequent calls! The time for OptiCaller to intercept the outgoing call, extract the callee's number adds only a small delay to the Call Back, this process and delay is not relevant to the Java client as the user has to input the callee's number directly into the Java client.

The Call Back call establish time includes the time required for connecting to the Internet (by the OptiCaller application), the time for the transaction between OptiCaller and the DoD server, and the call establish time for two calls from the IP-PBX on the server side (one for the call back to the caller, and the other for the call to the callee). According to the first test, the time to establish a call is around 7.0 seconds. However, the IP-PBX actually takes less time than the mobile phone to establish a call since it goes through the IP trunk. And the time for the transaction between OptiCaller and the DoD server may be different if another data connection, such as WLAN, were used. Note that we have excluded the time for the transaction between the DoD server and the IP-PBX because it is very small.

The Call Through call establishment time using OptiCaller is a little shorter than the Call Through call establishment time using the Java client both on the Nokia N95 and Nokia E61. This delay includes the

time for establishing two calls (one from the caller to the IP-PBX, and the other from the IP-PBX to the callee), and the time for transmitting the DTMF and decoding tones. The Call Through call establishment time difference between the OptiCaller and the Java client is due to the difference in the time that it takes each of the application to send the DTMF tones. (This time also depends on the specific mobile phone is used.) The time for sending the DTMF tones is approximately 5.5 to 6.0 seconds. (The sub test results are included in Table 22 in Appendix A). This seems to be a key bottleneck for establishing a call via Call Through.

12.1.3 Power Consumption

The OptiCaller application is mostly running as a background application to intercept the outgoing calls and to allow users to make calls using an alternative calling method. This means that the application is running all of the time in order to monitor the telephony status. Additionally, the OptiCaller application needs to monitor the incoming SMS messages looking for provisioning. They may lead excessive power consumption causing a noticeable reduction in talk and/or standby time for the user. Therefore, it was necessary to measure the battery life effects due to running the OptiCaller application.

12.1.3.1 Test Scenario

The test scenario consists of one mobile phone into which we have installed Test Application 2. This test application is used to learn the battery power costs of different operations. The available battery power level is divided into seven steps (which is a standard feature of Symbian OS based mobile phones). When the available battery power level changes, the kernel will publish a notification. Applications subscribing to this notification will be notified. This test application is such an application. When the test application receives one of these notifications it records the current time.

As a background application, we have focused on the effects on the battery life caused by the OptiCaller during the stand by time. As this time is much longer than the talk time and the energy consumed by the application was expect to be much less than the power consumed when talking. Two tests were done: 1. record the battery life of the mobile phone with OptiCaller installed and running, and 2. record the battery life of the mobile phone without OptiCaller. Both tests are started when the battery of the mobile phone fully charged (i.e., at level 7).

12.1.3.2 Test Results and Analysis

Table 17 illustrates the stand by time for each battery level of the Nokia N95 with and without OptiCaller running. The duration for each level was calculated from the time of the start of the next lower level minus the time of start of the higher level. However, as mentioned before, the battery level has seven steps, hence the test data does not include the duration for level 1 since end time of level 1 is also the start time of level 0. Unfortunately, once the battery level reduces to level 0, the mobile phone will shutdown automatically. Thus the exact start time of level 0 cannot be obtained. Fortunately, the data from level 7 to 2 is sufficient to estimate the power used by the OptiCaller application.

Table 17: Nokia N95 Battery Life

Battery Level	7	6	5	4	3	2	Sum
With	58h	15h38m	17h53m	17h49	20h45	4h49	134h54m
Without	58h55m	15h13m	18h32m	17h50	20h52	5h14	136h36m

According to Table 17, when OptiCaller is running its power consumption is roughly equivalent to 2 hours of standby time. Figure 30 shows the decrease in battery level on a Nokia N95 with the elapsed time. The horizontal axis is the elapsed time, which is from 0 to 8400 minutes (which is 140 hours), while the vertical axis is the battery level (from 7 to 1). Using this figure and the previous table we can see that the battery life difference is equivalent to 2 hours of standby time and is less than 2% of the whole battery capacity. Therefore the effect of running the OptiCaller application in the background is small, in fact sufficiently small that few users will notice the effect in practice.

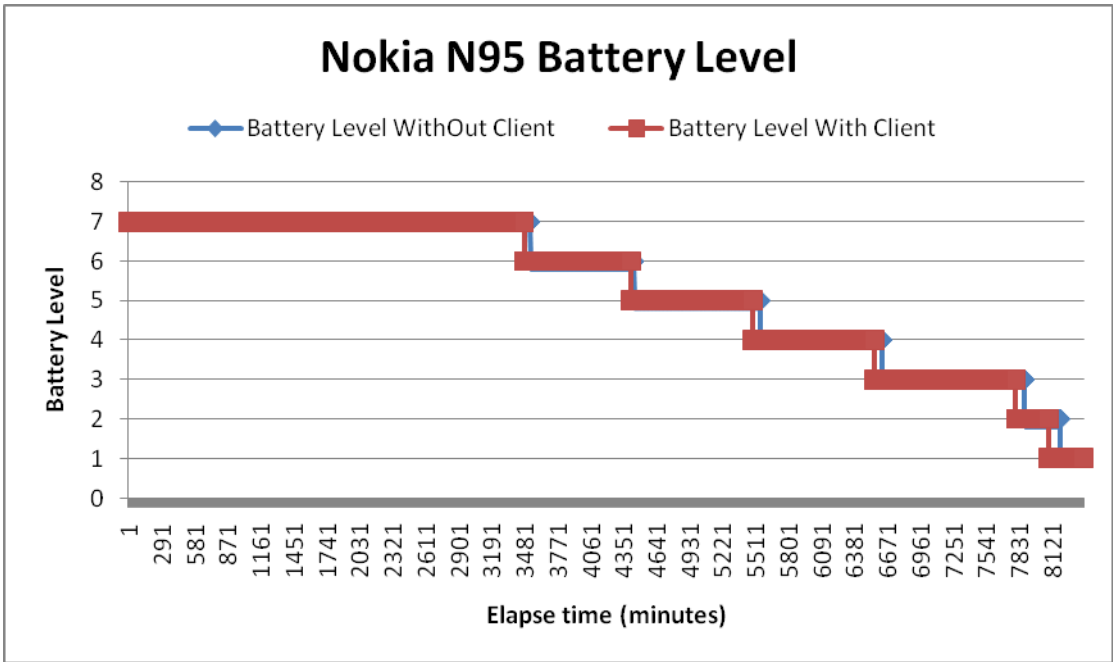

Figure 30: Nokia N95 Battery Level

A similar test was done for E61 module. Table 18 illustrates the stand by time for each battery level of Nokia E61 module with and without OptiCaller running. Figure 31 shows the battery level of Nokia E61 as a function of the elapsed time from 0 to 9900 minutes (which is 165 hours). According to these results, the battery life difference is also roughly about 2 hours of standby time on a Nokia E61. This is similar to that of the Nokia N95. In conclusion we can see that in neither case is the power consumption of the application significant.

Table 18: Nokia E61 Battery Life

Battery Level	**7**	**6**	**5**	**4**	**3**	**2**	**Sum**
With	68h12m	21h48m	20h32m	21h47m	19h41m	6h58m	158h58m
Without	68h56m	22h17m	21h5m	21h28m	20h13m	7h11m	161h10m

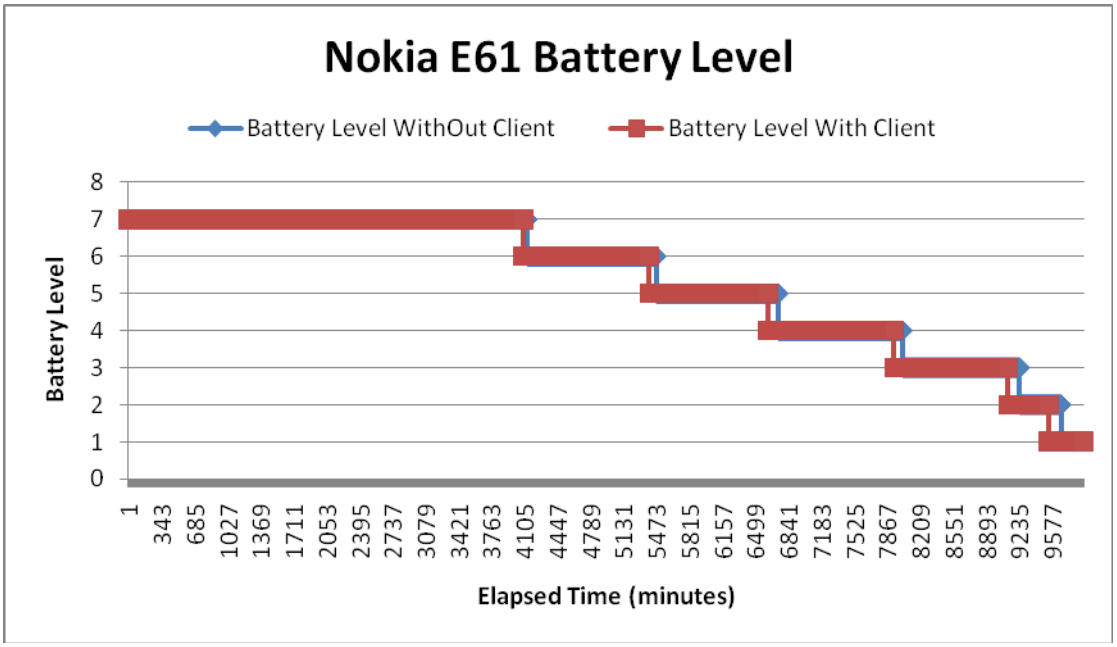

Figure 31: Nokia E61 Battery Level

12.1.3.3 Limitations

What is noticeable is that the stand-by times measured in the tests are very different from those stated in the device's profile. The information specified in the datasheet concerns the battery life for a new mobile phone. However, the two phones that were used in the test had been used for more than two years. With use, the battery lifetime of a mobile phone deceases. Further experiment utilized AT commands [37] to query the battery level. For the two phones used in the test, the percentage of battery capacity reported when the phone claimed to be fully charged were only 90% as reported by the AT command query, indicating that the battery could not be fully charged.

There is a risk of error in this approach of connecting the phone to a PC via a USB cable and sending the phone an AT command to determine the battery charge level, this is because when the phone is connected to the USB master it will attempt to use the USB master's power to recharge the phone's battery.

However, the test goal was to understand the power consumption of running the OptiCaller application, thus it does not matter if the battery life is the same as stated on the datasheet or not. The result is that the difference in battery lifetime when running OptiCaller is less than 2% of the standby time.

12.1.4 Conclusions From These Two Sets of Tests

According to the first test, normally, users need to wait for approximately 7 seconds to establish a call, while using OptiCaller takes three times as long to establish a call using Call Back and Call Through. However, the user does not care whether the additional time is a factor of 3 or 0.3, what they care about is the value of waiting longer - i.e., a longer waits leads to lower costs, hence the user has to consider the value of their time. Before we do this we will consider the some additional details of the call setup delays.

For Call Back, the additional time is due to the time required to connect to the Access Point, the time for the transaction between the application and the DoD server, and the time required to make the return call

from the IP-PBX to the caller. As we noted before the performance of OptiCaller is worse than the Java client, because the Java application keeps the connection alive after the first use of Call Back. We also noted that the time for connecting to the Internet and to perform the transaction can differ depending on the mobile phone, the operator, and the type of connection (such as GPRS, 3G, and WLAN) used. To improve the performance of Call Back, OptiCaller could keep the GPRS connection alive, just as the Java client does. However, this may incur an extra charge depending on the subscription and the operator.

For Call Through, the additional time due to the time required to transmit transmitting the DTMF tones and to establish the second call from the IP-PBX to the callee. Here we saw that the performance of the OptiCaller was better than the Java client, due to the difference in the time required to send the DTMF tones by the Symbian application versus the Java application. We also saw that the speed of sending the DTMF tones was the key bottleneck for Call Through when using OptiCaller.

Considering the length of a typical business call, the extra time required for call establishment might seem acceptable. However, if one does a cost estimate of this time we can see that for some users this might be acceptable - while for others it might not be. Consider an employee whose total cost to the company (for salary) is 30,000 SEK per month; then if this employee works 140 hours per month, then the cost of the employee is ~0.06 SEK/second. So an additional delay of 12 to 17 seconds at 140 calls per month would be a cost of 100 to 143 SEK. However, if the employee makes 10 calls per hour on average instead of 1 call per hour then these costs increase by a factor of 10! Even worse is that it is not simply the cost of the employee's time by the value of this time to the profits of the firm, hence the effect of this additional delay when multiplied by the number of employees could be completely unacceptable. Additionally from a human perception of time to complete an operation, the human user will become frustrated when the delay for something that should take a short period of time is longer than 2 seconds (consider the wait times that users expect when web surfing). We will return to the issue of the call establishment delay in the next chapter.

Unlike the results of the call establishment tests, the measurements of the power consumption of the OptiCaller application would appear to indicate that the additional power consumption by this application will not be noticeable by a typical user. We will also return to this issue in the next chapter.

12.2 Provisioning Server Evaluation

Chapter 11 introduced the OptiCaller Provisioning System. As a commercial product, the performance of the system needs to be evaluated in addition to its functionality. The performance test for the provisioning system focuses on the server side, since the server has to support the aggregated load caused by each of the clients. We will consider the costs to the individual clients in Section 12.2.3.

‘Load testing’ is one of the most important tests for a server, especially for web servers. [49] In ‘load testing’, the server is tested while being accessed by multiple users concurrently. Based upon the results of this testing the capacity of the server can be estimated. In this project, the capability of the OptiCaller provisioning server was tested; in order to estimate the number of users that can be supported by the server simultaneously.

As previous chapter explained, there are three types of users of the OptiCaller Provisioning System: 1. administrator, 2. manager, and 3. end user. The administrator and the managers perform administrative

operations such as add/update/delete groups. The ‘add’ and ‘delete’ group operations by either of these types of users will not lead to too much load (as these are basically simple database operations local to the server). However, once a group configuration is updated, the server will send an SMS message to each of the end users in this group indicating that there is an update - hence all the end users in this group will send requests to fetch the updated configuration. Normally, there are hundreds of end users per group, which means that compared to the load caused by the administrator and the managers, the load caused by the end users fetching configurations will be the greatest load for the provisioning system. Therefore in the following tests, only the performance due of the end users fetching the configuration is tested. The load caused by end users downloading the OptiCaller application has not been considered because it only depends on the JBoss server and the hardware of the test machine rather than the web application quality. It remains for future work.

12.2.1 Test Equipment

To test the provisioning system we have used two machines, with one acting as the OptiCaller Provisioning System server (Test Machine A) and the other (Test Machine B) acting as one or more clients. The details of these two machines and the software that we have used are given below.

- Test Machine A
  - Module: Dell System PowerEdge SC420
  - CPU: Intel Celeron 2.53GHz
  - Memory: 1GB
  - Hard Disk Drive: Hitachi Deskstar 7K400 400GB (SATA 1.5Gb/s)
  - Operating System: Fedora 9 Kernel Linux 2.6.27
- Test Machine B
  - Module: Dell System 8400
  - CPU: Intel Pentium 2.8GHz
  - Memory: 1GB
  - Hard Disk Drive: Hitachi Deskstar 7K250 80GB (SATA 1.5Gb/s)
  - Operating System: Windows XP Professional Service Pack 3
- Test Tools
  - HP LoadRunner 8.1 Feature Patch 4
  - Apache JMeter 2.3.4
- NAT Router: Linksys® RV082 10/100 8-Port VPN Router
- Internet Service Provider (ISP): Tele2

12.2.2 Test Tool Introduction

12.2.2.1 LoadRunner

LoadRunner is a testing product developed by HP. [50] It is used for examining system behavior and performance by generating actual load on a system. This software can emulate hundreds of concurrent users who access the application simultaneously, just as in real-life. These test results will be analyzed to understand the system's behavior.

LoadRunner consists of three parts: ‘Virtual User Generator’, ‘Controller’ and ‘Analysis’.

Virtual User Generator A tester records the actions against the application to be tested as a script,

then verifies the script. Additionally, this software allows the tester to make modifications to the scripts to address, Error Handling, Correlation, and Parameterization of the script.

Controller The controller is the actual load generator. Testers can use the controller to emulate hundreds or thousands users simultaneously using the script generated by Virtual User Generator. Scenario settings need to be done; these describe what script will run, when it will run, and how many virtual-users will run, and so on. The controller has various monitors to show the performance of different aspect of the system, such as the number of virtual-users, the number of connections, the average response time, and so on. The specific settings for our testing will be described in Section 12.2.3.1.

Analysis The completed test scenario results can be viewed using the analysis part of LoadRunner. This includes generating suitable graphics and various means of processing the raw data.

12.2.2.2 Apache JMeter

Apache JMeter is an open source Java program designed to load test and measure performance of web servers. [51] It was originally designed for testing Web applications, but it supports other tests, such testing JDBC database connections, LDAP, and Mail - POP3(S) and IMAP(S), and so on. It supports a variety of reports for helping analyze the server's performance.

There are several components available for testing: ‘Threads Groups’, ‘Configuration Elements’, ‘Samplers’, ‘Listeners’, ‘Logic Controllers’, ‘Assertions’, ‘Timers’, ‘Pre Processors’, ‘Post Processors’, ‘Miscellaneous Features’, and ‘Reports’. Only the first four components were used in our testing. A brief description of these four components is given below.

Threads Group A thread group is a collection of threads. Each thread can be regarded as an end-user. The other components run in a thread (consider as a scope in Jmeter).

Samplers Samplers performs the actual work of JMeter. Each sampler generates one or more sample results. These sample results have various attributes, such as success/fail, elapsed time, data size, and so on. These samples can be viewed using the various ‘Listeners’. The list of existing samples includes ‘JDBC Request’, ‘LDAP (Lightweight Directory Access Protocol) Request’, ‘HTTP Request’, and so on.

Configuration Elements Configuration elements can be used to set up configuraions and variables for later use by samplers. These configuration elements are processed at the start of the scope, before any samplers in the same scope for example. It includes ‘JDBC Connection Configuration’, ‘LDAP Request Defaults’, and ‘HTTP Request Defaults’ and so on.

Listeners Listeners are used for ‘listening’ to the test results. Listeners also support viewing, saving, and reading saved result files. The results can be saved in XML format. These results are processed at the end of the scope. Available listeners include ‘Aggregate Report’, ‘Graphic Results’, ‘Summary Report’, and so on.

12.2.3 Web Server Test

As the OptiCaller provisioning server is implemented as a web service, we will test it as one might test any other web service.

12.2.3.1 Test Scenario

Figure 32 shows the test scenario. The OptiCaller provisioning server runs on the Test Machine A. Test Machine A is connected to Internet with a public IP address. The LoadRunner is used as the 'load testing' tool which runs on the Test Machine B. Test Machine B is behind a NAT router on a private IP network. The NAT router is connected to the same Internet Service Provider (ISP) network as Test Machine A is connected to (see Section 12.2.1).

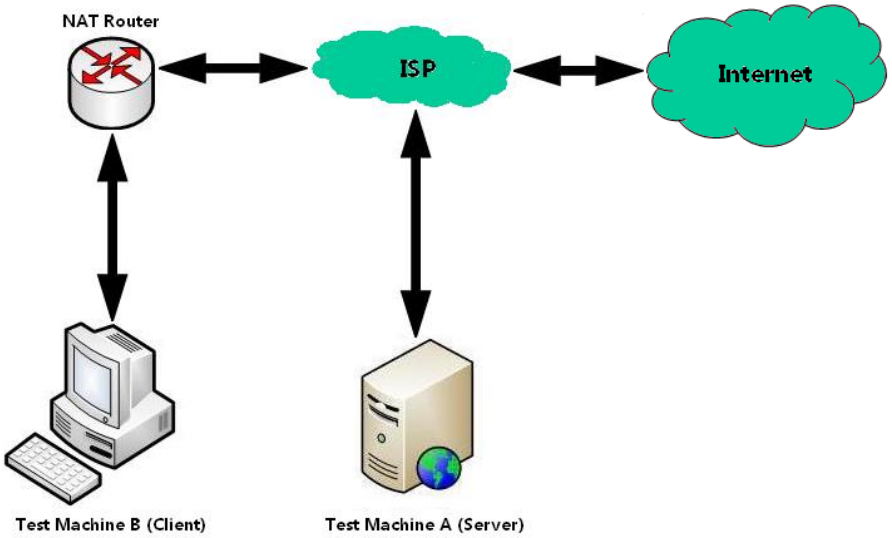

Figure 32: Provisioning Server Test Scenario

First, a test script is recorded based on a single end user accessing the provisioning server. The content of script will later be used to emulate an end user sending a request to the server for updating. This script needs to be verified to ensure that the script will be suitable for subsequent use. After script verification, the LoadRunner Controller can use this script to simulate end user behavior by any number of end users.

There are four settings for the LoadRunner:

Quantity The quantity is the number of virtual-users the LoadRunner will emulate.

Initialize all Vusers before Run LoadRunner will initialize all virtual-users based upon this setting before starting to run the first virtual-user.

Ramp Up - Load all Vusers simultaneously LoadRunner will load and initialize all of the virtual-users and run the script for them simultaneously.

Duration The duration setting determines how long the scenario continues running after finishing the ramp up. The duration can be 'Run until completion' (run the script once for each virtual-user), 'Run for' a specific time, or 'Run indefinitely'.

During the test, we started with 'Quantity' set to 100, and increase this value by 100 for each round of testing. Once the number of virtual users exceeds the capacity of server, the virtual-users number will be decreased by 50 to estimate the actual capacity of the server. The 'Duration' was set to 1 minute 30 seconds as this generates enough data to calculate the performance of the server with different numbers of virtual-users. The other two options were set to cause the number of virtual-users and server to reach a steady state quickly. To avoid the effects of database caching, the username and password are gotten from a parameter table in the LoadRunner.

In addition to configuring the settings for LoadRunner, there are some other settings that need to be adjusted for the test. Some of these settings concern the configuration of the operating system and some of them concern the configuration of the JBoss server itself. First we will consider the OS configuration changes, then the JBoss server changes.

Firstly we must increase the number of file descriptors that can be open simultaneously. In the Linux operating system, file descriptors can refer to files, directories, sockets, and so on. If we attempt to open more sockets than the file descriptor limit, then the web request will fail and the server will throw the exception 'java.net.SocketException: Too many open files'. The default file descriptor number limitation is 1024. The command 'ulimit –n' can be used to get and set this value. We set the value to 1000 for testing.

Secondly, the number of threads the JBoss server processes is too limited. The setting for this value in JBoss is defined in the file 'server.xml'. There are three parameters that need to be adjusted: 'maxThreads', 'maxSpareThreads', and 'minSpareThreads'. [33]

maxThreads This parameter limits the maximum number of request processing threads that can be created. This value determines the maximum number of simultaneous requests that can be handled. The default value is 200.

minSpareThreads This is the number of request processing threads that will be created when the server starts. The default value is 4.

maxSpareThreads Once the number of threads excesses the 'minSpareThreads', the server always keeps 'maxSpareThreads' waiting idle. However, the number of threads cannot exceed the 'maxThreads' value. The default value is 20.

Each of the above settings affects the performance of the server. Normally, the 'maxThreads' is 25% more than the maximum expected load (i.e., the maximum number of concurrent requests), the 'minSpareThreads' should be slightly more than the normal load, and the 'maxSpareThreads' should be a little more than the peak load. In our tests these settings were adjusted according to the different test cases.

Another setting is 'acceptCount' which is the maximum queue length for incoming connection requests when all possible request processing threads are in use. This parameter is also set in 'server.xml'. Requests received when the queue is full will be refused, in which case, the LoadRunner will get the error: 'Connection refused'. The default value is 10. In all test cases, the value for 'acceptCount' was set to 1000 so that the queue length would not be the performance bottleneck.

As a web database application, the configurations for the connection with the database also need to be adjusted. In this project, the database server is MySQL server. There are three configurations parameters relevant to connection with the MySQL server: 'max-pool-size', 'min-pool-size', 'idle-timeout-minutes', and 'max-connections'.

The first three settings are particularly relevant for the JBoss connection pool. The 'max-pool-size' is the maximum number of connections that can be maintained to the database – the default value is 20, while 'the min-pool-size' is the minimum number of connections maintained to the database - the default value is 0. If a connection is not used after the time defined as the 'idle-timeout-minutes', then the connection will be closed and removed. The default value is 0 - meaning that the idle-timeout is infinite, i.e., idle

connections are never closed. These settings are configured in the file ‘mysql-ds.xml’.

The ‘max-connections’ parameter is a setting for the MySQL server, this sets the maximum number of connections that this instance of the MySQL server can accepted. The default value is 100. This can be modified by using the command ‘set GLOBAL max_connections=1000’ to set the parameter to 1000.

All of the above settings affect the performance of the connections between the JBoss and the MySQL server. The settings for the JBoss connection-pool size are adjusted depending on the test cases; while the ‘max-connections’ for the MySQL server was set to a fixed value of 1000. The value of 1000 was chosen as it is greater that the total number of virtual users that we will consider.

12.2.3.2 Test Results

As mentioned in previous section, the test starts with 100 virtual-users. Figure 33 shows the number of virtual-users as a function of time. According to the figure, the number of simultaneous virtual-users reaches 100 at 2 seconds after the start of test scenario, this number of virtual users remains at 100, by starts to fall after 1 minute and 22 seconds (i.e., 1:32). This is exactly as expected, since we set the duration of the testing to be one minute and thirty seconds after the ramp up (which took 2 seconds).

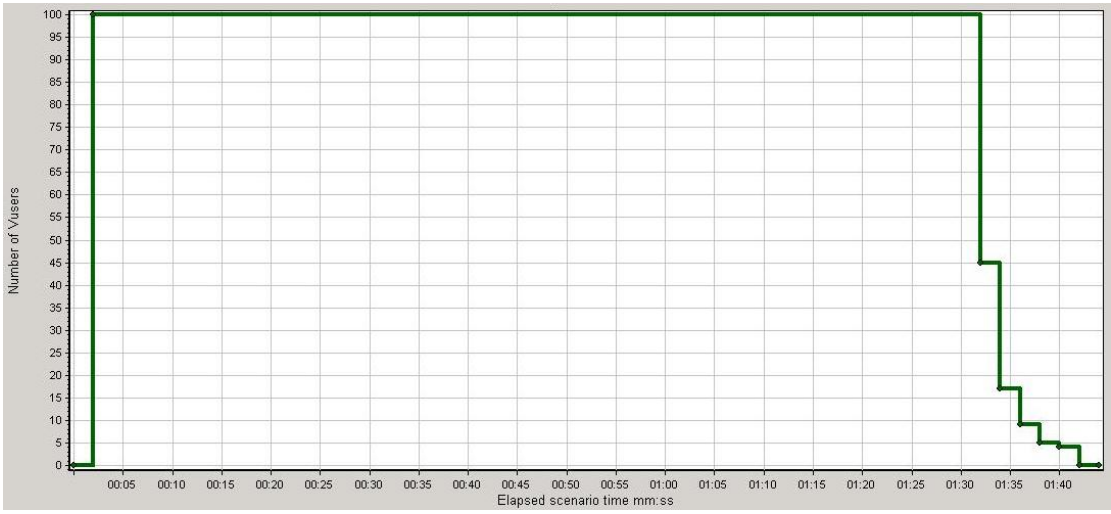

Figure 33: Number of Runing Virtual Users

However, simply looking at the number of virtual-users as a function of time does not indicate the load generated by LoadRunner on the server. Instead, we will examine the number of simultaneous connection to understand the load. Figure 34 shows the number of connections as a function of the elapsed time. We can see that the number of connections reaches a peak after 6 seconds, and remains at 100 connections, then starts to reduce at 01:32. There is a small reduction from 100 to 99 at approximate 00:45 which can be safely ignored (as a sampling error of the emulation - i.e., there were a sampling time when there was an available connection that had not been assigned to a virtual user's request).

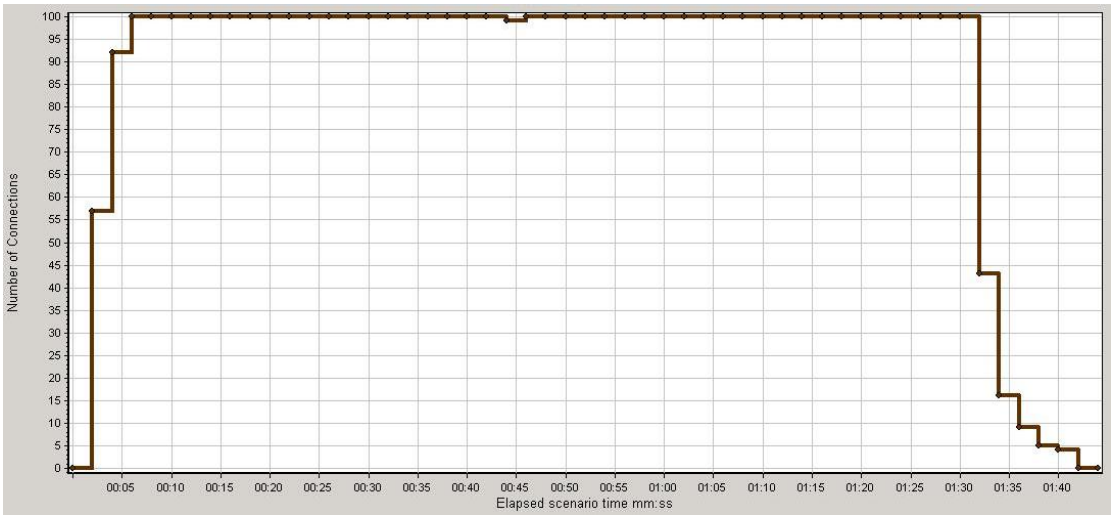

Figure 34: Number of Connections

Figure 35 illustrates the number of the new connections and the number of closed connections per second. This figure more clearly shows the connection status. The red curve represents the new connections while the green curve represents the closed connections. It is clear that after 6 seconds, the red curve and the green curve merge to the same curve, which means that the number of the new connections equals to that of the closed connections, until 01:30. During this period, the number of connections remains steady at 100 connections as shown in Figure 34. We can see that just before 00:45, there is a small gap between the red curve and the green curve, which leads the small fluctuation on the curve shown in Figure 34.

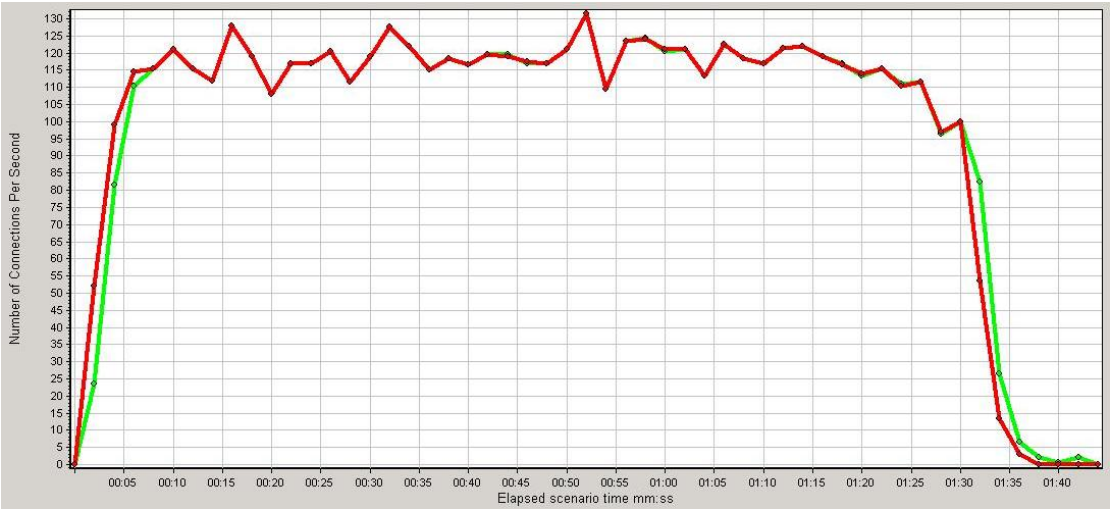

Figure 35: Number of Connections Per Second (The red (upper curve at the start) show the number of new connections during the last second and the green curve shows the number of closed connection during the last second.)

The goal of this testing is calculating the average respond time of a transaction at full load, so only the period during which the maximum number of connections are in use (we refer to this as the ‘used period’) are of interest to us. In the case of 100 virtual-users, the ‘used period’ was from 00:10 to 01:30.

The same test was repeated for 200, 300, 400, 500, and 600 virtual-users. Because it takes LoadRunner different amounts of time to initiates the maximum number of connections, the start time of the ‘used

period' will be different ; however, the ending time will always be one minute and thirty seconds after the system load has ramped up - as this specified as a parameter to the test runs. To standardize and simplify our analysis test cases, the 'used period' was defined as the 1 minute following when the maximum number connections are established. Additionally, during the 'used period', the number of connections was not allowed to fluctuate by more than 1%. If the fluctuation was greater than this we repeated the test run. Fortunately, we did not repeat the test for this reason during the testing.

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. According to reference [52], the response time for a Web page should not be more than 3 seconds on average, and never more than 5 seconds. Thus, we assumes that if the response time for pages is within 2 seconds, then the user's perception is that the system is perfect, between 2 and 5 seconds is considered normal, but having to wait more than 5 seconds is bad. Table 19 illustrates the test results from 100 to 500 virtual-users. These results include the average response time for all the transactions during the 'used period', and the percentages of the transactions about which the users feel pleased, normal, and bad.

Table 19: Provisioning Server Test Result

User Number	Average Time	Perfect	Normal	Bad
100	0.837	89.50%	9.50%	1.00%
200	1.670	69.30%	27.70%	3.00%
300	2.482	43.00%	50.70%	6.30%
400	3.360	33.70%	56.10%	10.20%
450	3.869	30.60%	53.70%	15.70%
500	4.407	21.60%	53.20%	25.30%

12.2.3.3 Analysis of the Test Results

According to the Table 19, with an increasing number of users, the average response time increase and the percentage of perfect transactions decrease, while the percentage of less than perfect transactions increases. The average time is always acceptable even when the number of users is 500, as 4.4 seconds is still in the range of 'normal'. However, this average is very misleading as can be seen by the fact that only 21% of the users would be getting 2 second or less response, while more than 25% of users would be experiencing excessively long delays.

As we can see in the table, when the number of users increases from 400 to 500, the percentage of 'bad' transactions rises dramatically from 10.20% to 25.30% while the percentage of 'perfect' transactions decreases from 33.70% to 21.60% which is less than that of 'bad' transactions. It means the server's performance is becoming unstable even though the average response time is acceptable. This indicates that the maximum number of users that the system can support is less than 500. Hence we made another test run with 450 virtual-users. We chose this value as it is the middle between the case of 400 virtual-users and that of 500 virtual-users. With 450 users we can see that the percentage of 'perfect' transactions is double the percentage of 'bad' transactions. However, at this point we can see that the system is coming under pressure since the percentage of perfect experiences decrease by ~3%, but the percentage of bad experiences increased by ~5.5% in comparison the performance with 400 users. It is interesting to note that the percentage of normal experiences also decreased by 2.4%.

Based upon these tests we estimate that the web server can support between 400 to 450 users with an acceptable performance, while actually supporting 100 users simultaneously.

12.2.4 JDBC & MySQL Test

In the provisioning system, there are a lot of database operations, such as querying the database to validate users (administrator, managers, and end-users), storing the group configuration into the database, and so on. The provisioning server uses JDBC to communicate with the database. We ran a number of tests when using JDBC to access the MySQL database to see if these lead to the performance bottleneck of the provisioning server.

12.2.4.1 Test Scenario

The MySQL database server runs on the same machine as the provisioning server. So this test uses only Test Machine A. Since LoadRunner 8.1 does not support testing MySQL, JMeter was used for this test instead of LoadRunner.

For the MySQL server, the 'max-connections' parameter was set to 1000 as this was viewed as being safely above the number of simultaneous connections required to serve the maximum number of users that we were concerned with.

In JMeter, a 'Thread Group' needs to be added to the 'Test Plan' first. There three settings relevant to the 'Thread Group': 'Number of Threads', 'Ramp-Up Period', and 'Loop Count'. The 'Number of Threads' is similar to the number of virtual-users in the LoadRunner. According to the previous section, the provisioning server can support 400 to 450 simultaneous users. So the 'Number of Threads' was set to 450 in this test. The 'Ramp-Up Period' is the length of period the JMeter to open the threads. This was set to 30 seconds. However, unlike LoadRunner, JMeter cannot initiate all the threads before using them to during the test. Instead it opens new threads while the previously opened threads are sending test requests to the database, which means the JMeter will take longer time to open all the threads than LoadRunner took. The 'Loop Count' is the number of operations loop iterations that each thread will do. It is set to 'Forever', hence the test must be stopped manually. The reason for setting the Loop Counter to this value is that once the thread finishes its operation, it will be closed. In this test, we wanted to make sure that no threads were closed before all the threads are opened.

Next an element called 'JDBC Connection Configuration' needs to be added in the 'Thread Group'. The setting for the 'JDBC Connection Configuration' is similar to the setting for the JBoss connection pool in the file 'mysql-ds.xml', including the 'Database URL', 'JDBC Driver class', 'Username', 'Password', and 'Max Number of Connections' (which were set to the same as the 'max-pool-size' for JBoss connection pool). The 'Max Number of Connections' was set as 500, while the other settings are set to the same values as the settings for the JBoss connection pool.

Finally, an element called 'JDBC Request' is added in the 'Thread Group'. The test SQL query statements are set in this element. In this test, the database operations include verifying the username and password and fetching the corresponding configurations.

Additionally, various 'Listener's are added to get collect and analyze the results, these included 'Summary Report', 'Aggregate Table', and 'Result Table'. To avoid database caching, the pair of the username and password is fetched from the parameter table 'User Defined Variables'.

12.2.4.2 Test Results

Figure 36 indicates the number of threads started as a function of the elapsed time. According to this figure, it takes two minutes for the JMeter to open all the threads (450). Then the 450 threads keep sending a query to database simultaneously. This full load lasts for about two and half minutes. We considered only transactions during the full load situation. In the end JMeter takes 15 seconds to stop all the threads.

According to the summary report generated by the JMeter, there were 41862 requests sent from the JMeter to the database during the entire test period. All of these requests got the successful responses. The average response time for the all requests was 0.383 seconds with a variance of 1.427 seconds. Of these requests, approximately 26525 requests were sent from JMeter to the database during the full load period. The average response time of the requests during the full load period with 450 threads was 0.473 seconds with a variance of 1.156 seconds.

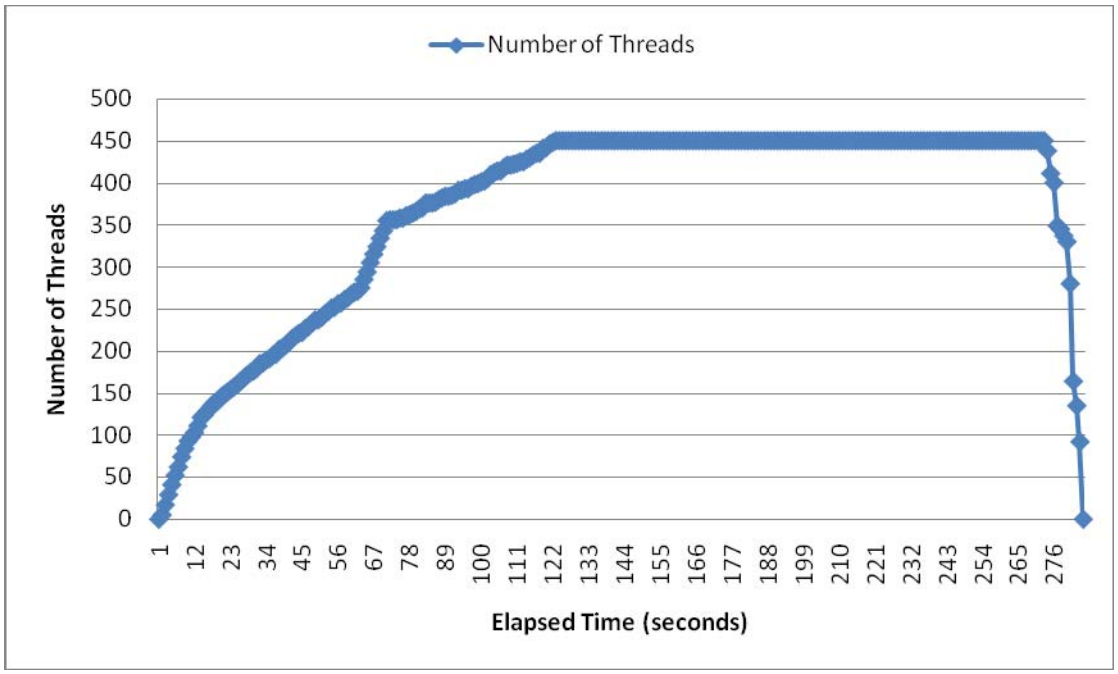

Figure 36: Number of Threads as a function of Time

12.2.4.3 Analysis of Results

As mentioned before, JMeter cannot initiate all the threads before starting to send the requests. Thus roughly 15337 requests (41862-26525) were completed before all the threads are started. In our analysis only the 26525 requests sent after all the threads were started are used.

As stated in the prior section the average response time of the requests during the full load period is 0.473±0.407 seconds. The time for the communication between JBoss and the database (the MySQL server) using JDBC only occupies 12.23% of the total time consumed by all of the server operations. Hence JDBC (and the MySQL database) is not a major performance bottleneck for the provisioning server.

12.2.5 Conclusions

According to our test the current OptiCaller provisioning server can supports between 400 to 450 users. The peak load will happen when there is a configuration update for a group of end-users. Normally, in the DoD solution, each group includes at most 300 to 400 end-users. Additionally, it may take some time for all end-users to receive the provisioning SMS, thus further spreading the load over time. Thus it seems that the OptiCaller provisioning server will be capable of handling the load caused by an update to a single group of less than 450 users using a server comparable to Test Machine A. However, the system administrator and managers should be aware that this is the maximum performance (for this platform), thus they should rate limit their updates to the provisioning system to stay below this bound.

As a web server, the performance of the OptiCaller provisioning server is determined by the application code efficiency, the database capacity, the network status, and the hardware of machine on which the application runs. Further work should investigate what happens if there are changes in some of these parameters, especially a change in the configuration of the hardware of the server.

According to the second test results, the time consumed by communication between the application and the database, JDBC and the MySQL server are not the key performance bottleneck for the provisioning server.

Currently, the OptiCaller provisioning server runs on the simple desktop computer as a test server. Upgrading the hardware or running the application on a server c class machine could improve the server's performance. However, quantifying such an improvement remains for future work.

Building the OptiCaller Provisioning System as a distributed system is another way to improve the provisioning system's performance. Using a load balance mechanism could potentially increase the system's capacity. However, quantifying this also remains for future work.

13. Conclusions and Future Work

After summarizing all the work done in this project, including the research and the development done, this chapter presents some conclusions. Following this a summary of future work is presented.

13.1 Conclusions

The conclusions can be divided into several parts. Conclusions about OptiCaller as an application and conclusions about the provisioning system

13.1.1 OptiCaller

A mobile extension application running on Symbian OS based mobile phones, called 'OptiCaller', was designed, implemented, and evaluated in this project. OptiCaller has the following features (except for the HTTP/HTTPS call-back function, these are new functions as compared to the earlier Java DoD client):

- It supports two cost-saving call methods: call-back, and call-through.
- It supplies two ways to initiate the call-back service: HTTP/HTTPS and SMS.
- It provides the user with an alternative means to initiate calls without requiring a change in the user's dialing behavior.
- It supports several Mobile Extension (MEX) functions, such as setting presence status, joining groups, transferring calls, and so on.
- It is compatible with most of PBX solutions; using dynamic configurations for each function.
- It is the OptiCaller Provisioning System client side which supports the function to keep updating with the OptiCaller provisioning server.

OptiCaller was evaluated with respect to the time required to establish a call and the power consumption of the applications.

OptiCaller takes about three times as long to establish an alternative call as a direct (normal) call. Compared to the new Java client, it takes about five more seconds for OptiCaller to make a call-back call because OptiCaller does not maintain a data connection. While OptiCaller takes slightly less time than the Java client to make a call-through call. However, comparing the performance of these two alternative ways of establishing a call to the normal direct approach with regard to the total duration of an average business call, the extra delay in call establishment time seems acceptable. However, we noted in the previous chapter that a deeper analysis of costs and benefits needs to be made - as it is not clear that the extra delay would be acceptable if the user makes many short calls. Such an economic analysis lies outside the scope of this report.

With regard to the power consumption caused by running this application, OptiCaller only reduced the test mobile phone's battery life by about two hours (in terms of standby time). Considering that these mobile phones can stand by for several days, the difference in battery life is not of practical importance.

13.1.2 OptiCaller Provisioning System

The configurations for the OptiCaller are complicated because it is designed to work with several different PBX solutions. To handling the deployment and configuration updates for the OptiCaller application, a provisioning system, called 'OptiCaller Provisioning System', was designed, implemented, and evaluated in this project. This provisioning system provides the administrators (including the super administrator and managers) a platform to manage the OptiCaller application as deployed on the end-users' mobile phones.

The OptiCaller provisioning server was tested in order to estimate its load capacity. According to our test, the server can support between 400 to 450 end-users for an update. The scaling of this system seems to be sufficient to support the typical number of end users in a group using a desktop computer as a server. Further work is needed to understand what the proper configuration of such a server should be.

13.2 Future Work

13.2.1 OptiCaller

OptiCaller attempts to be user friendly, but there are some improvements needed, both for Call Back and Call Through.

For Call Back, the call establishment time is three times the normal direct call establishment time. One component of this delay that could be improved is to reduce the call-back call establishment time. One method of doing this is to like the new Java client keep the data connection active after the first use, thus subsequent call-back calls only take double time of the normal direct call. However, there are several potential drawbacks to this solution: (1) increased costs due to maintaining the data connection, (2) increased battery power consumption due to keeping this connection active, and (3) concerns of the user that their device is connected to the Internet for a longer period of time - especially when they do not have any active data communications (that they are consciously aware of).

For Call Through, the call establishment time is triple that of a normal direct call establishment. Currently, OptiCaller uses the Symbian Telephony Services (see Section 7.2.1) to send the DTMF tones. The speed for transmitting the DTMF is very slow (about two tones per second), and this speed is not adjustable. There is an API called CTonePlayer [53] in the Symbian OS Library, this could be used for sending DTMF tones, and the speed can be adjusted. The speed can be raised to more than two tones per second. However, the DTMF tones need to be decoded at the PBX. There may be some DTMF speed limitations on some PBX solutions. To avoid a conflict, a new configuration field can be introduced so the OptiCaller users can adjust the speed by themselves. Since MEX functions are also invoked by sending DTMF tones, the optimization of the speed of sending DTMF tones can be enjoyed by the MEX functions too. However, my thesis examiner has suggested several alternative methods of reducing this delay, by eliminating DTMF tones altogether; potentially eliminating this element of the delay. For instance, one could send the extra digits in the call to the IP-PBX using direct inward dialing (also known as direct dial-in) - thus avoiding any extra time to send these digits (if the total number of digits is below the limit for the country's maximum number of digits - the maximum number of digits for E.164 is 15 digits for international numbers). Thus if you dial a local number of ~10 digits, you could pass 5 digits to the IP-PBX. This could potentially decrease the additional delay of Dial Through time to simply the extra time required for the second call.

Evaluations for the SMS call-back remains to be done. It was not done during this project due to limited time and other resources. And evaluations for the DoD server and the IP-PBX (Asterisk in this project) remains to be done, such as the transaction time between DoD server and IP-PBX and the time for establishing call between IP-PBX and mobile phones. It will help to understand more about the call establishment delay by using OptiCaller. Additionally, it would be interesting to compare the performance when OptiCaller works with different solusions.

OptiCaller needs to provide more functions to be compatible with other PBX solutions and improve the performance, such as:

- Encryption on SMS call-back request. It would be better to send the SMS call-back request, which might contain the username and password, in a secure way.
- A DTMF call-back function, as this is the traditional way to make a call-back call.
- SMS MEX functions. Currently, the only way to use the MEX services is by sending the code as the DTMF tones. It would be better to provide alternative way to use these services. The DoD solution plans to implement MEX services whose code can be sent by both the DTMF tones and SMS. (Additional, methods should also be considered - see the DTMF discussion above.)
- Check configuration correction function. It would be better to check the correction and integrity of the configuration returned from the provisioning server, for example by using hash value.
- Web third party calls supporting. [54] Web third party call is the service provided by most VoIP providers which give the user ability to make a call via their web sites. It is similar to HTTP Call Back in DoD solution. The advantage is that the call is made by the VoIP provider directly. So the limitation of the software IP-PBX (Asterisk) [6] in the DoD solution can be avoided and OptiCaller could be used more widely.

13.2.2 OptiCaller Provisioning System

Based upon our evaluation, the OptiCaller provisioning server can supports between 400 to 450 end-users. However, it remains several interesting test:

The test for load from the user's downloading application installation file is skipped in this project. As mentioned before, it more depends on the JBoss server and the hardware class. It would be better to test the downloading load to understand the provisioning server better. Additionally, testing the performance difference when using different hardware class on the server machine remains as the future work.

The provisioning system could be improved both with respect to the code aspect and with respect to the system implementation, specifically:

- Enterprise JavaBeans (EJB) [32] is supported by the JBoss application server. EJB enables rapid and simplified development of distributed, transactional, secure, and portable applications based on Java technology. [32] The performance of the provisioning server can be improved if it were to use EJB. For instance, although the current provisioning server uses JPA to handle the JDBC transactions, the server needs to manage the entities by itself. However, by using EJB, the entities are managed by the EJB container in a more sophisticated way.

- Load balancing is another way to optimize the OptiCaller Provisioning System performance. The provisioning system could also be built as a distributed system instead of a single centralized server. In this way the total load could be distributed over several different servers potentially improving the system's capacity and avoiding a single point of failure.
- Using Internet based SMS messaging solutions makes sending the SMS provisioning messages much quicker and more stable. Additionally, it can avoid the problem introduced by the SMSLib library.

To make the provisioning system more useful and sophisticated, several functions are needed to be implemented in the future:

- It would be better to support the provisioning of the Java client. So the provisioning of the OptiCaller and the Java client can share the same provisioning platform.
- The algorithm for checking the integrity of the uploaded file needs to be introduced to the provisioning system, such as Message-Digest algorithm 5.
- Calling method selecting function would be very useful to users. It would be better that the provisioning server can calculate the cost depending on user's operator and subscription, and deploy the cheapest calling method for users. It needs to collect more users' information and various operators' information.

References

[1] OptiCall Solutions AB, September 2007, available at http://www.opticall.se

[2] Nortel Solution: Mobile Extension, 2008, available on http://www.nortel.com/solutions/cablemso/collateral/nn123798.pdf

[3] Sun Microsystems, Introduction to OTA Application Provisioning, November 2002, available at http://developers.sun.com/mobility/midp/articles/ota/

[4] Max Weltz, Dial over Data solution, Master's Thesis, Royal Institute of Technology (KTH), School of Information and Communication Technology, COS/CCS 2008-2, February 21 2008, available at http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080221-MaxWeltz_ExjobbReport-with-cover.pdf

[5] Sun Microsystems, J2ME Mobile Information Device Profile (MIDP); JSR 37, JSR 118 Overview, November 2003, available at http://java.sun.com/products/midp/overview.html

[6] Zhang Li, Service Improvements for a VoIP Provider, Master's thesis, Royal Institute of Technology, School of Information and Communication Technology, TRITA-ICT-EX-2009:104, August 2009, available at http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/090829-Zhang-Li-with-cover.pdf

[7] Ioannis Metaxas, not available yet

[8] The OnState Team, Skype PBX, June 2009, available at http://www.on-state.com/skype-pbx.html?_kk=skype%20gateways&_kt=5ef5e2da-727f-4248-8a87-c18a04d90297&gclid=CMaX9f-puZsCFdMWzAodE23wBg

[9] The Onstate Team, June 2009, available at http://www.on-state.com/

[10] Telepo, June 2009, available at http://www.telepo.com/home/

[11] Telepo Mobile+, June 2009, available at http://www.telepo.com/product-architecture/telepo-mobile-plus-/

[12] Richard Harrison and Mark Shackman, Symbian OS C++ for Mobile Phones, John Wiley & Sons Ltd., August 2007

[13] Andreas Jakl, Symbian OS Memory Management, May 2008, available at http://symbianresources.com/cgi-bin/schlabo/dl.pl?MemoryManagement

[14] Jo Stichbury, Symbian OS Explained - Effective C++ Programming For Smartphones, John Wiley & Sons Ltd., 2005

[15] Ben Morris, CActive and Friends, June 2008, available at http://developer.symbian.com/main/downloads/papers/CActiveAndFriends/CActiveAndFriends.pdf

[16] Aapo Haapanen, Active objects in Symbian OS, Master's Thesis, Department of Computer Sciences, University of Tampere, April 2008, available at http://www.cs.uta.fi/research/theses/masters/Haapanen_Aapo.pdf

[17] Andreas Jakl, Symbian OS Client-Server Framework, May 2008, available at http://symbianresources.com/cgi-bin/schlabo/dl.pl?ClientServer

[18] Ben Morris, The Symbian OS Architecture Sourcebook – Design and Evolution of a Mobile Phone OS, John Wiley & Sons Ltd., 2007

[19] Andreas Jakl, Symbian OS GUI Architectures, May 2008, available at http://symbianresources.com/cgi-bin/schlabo/dl.pl?Gui

[20] Nokia Corporation, Symbian OS View Architecture, July 2005, available at http://sw.nokia.com/id/3cab67ad-db01-400a-9467-91b8be7ccbba/Symbian_OS_View_Architecture_v1_1_en.pdf

[21] Nokia Corporation, C++ Developer's Library 1.4, 2008, available at http://www.forum.nokia.com/document/Cpp_Developers_Library

[22] Iain Campbell, Symbian OS Communications Programming, John Wiley & Sons Ltd., 2007

[23] Ben Morris, Platform Security and Symbian Signed: Foundation for a Secure Platform, Symbian Software Ltd, January 2008

[24] VeriSign, September 2009, available at http://www.verisign.com

[25] TrustCenter, September 2009, available at http://www.trustcenter.de/en/index.htm

[26] Symbian Software Ltd, A developer's guide to Symbian Signed, September 2005, available at https://www.symbiansigned.com/developerguidetoSymbianSigned.pdf

[27] ETel Core in C++ API reference, 2007, available at http://www.symbian.com/Developer/techlib/v70sdocs/doc_source/reference/cpp/ETelCore/index.html

[28] Symbian Software Ltd, Symbian OS v9.2 library, 2006, available at http://developer.symbian.com/main/documentation/sdl/symbian92

[29] Sun Microsystems, Java SE at a Glance, September 2009, available at http://java.sun.com/javase

[30] Douglas K. Barry, Service-oriented architecture (SOA) definition, September 2009, available at http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

[31] Sun Microsystems, Java EE at a Glance, September 2009, available at http://java.sun.com/javaee

[32] Sun Microsystems, Enterprise JavaBeans Technology/, September 2009, available at http://java.sun.com/products/ejb

[33] Marc Fluery, JBoss Admin Development Guide, September 2009, available at http://docs.jboss.org/jbossas/admindevel326/html/index.html

[34] Addison Wesley, Database Access with JDBC, July 2001, available at http://www.informit.com/articles/article.aspx?p=167843

[35] Sun Microsystems, Java SE Technologies - Database, September 2009, available at http://java.sun.com/javase/technologies/database/

[36] Sun Microsystems, DataSource, 2001, available at http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/datasource.html

[37] Wikipedia, Hayes command set, September 2009, available at http://en.wikipedia.org/wiki/Hayes_command_set

[38] T.Delenikas, SMSLib, September 2009, available at http://code.google.com/p/smslib/

[39] Nokia Corporation, Carbide.c++, September 2009, available at http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/IDEs/Carbide.c++

[40] Sun Microsystems, Java Persistence API, September 2009, available at http://java.sun.com/javaee/technologies/persistence.jsp

[41] The Eclipse Foundation, Eclipse homepage, September 2009, available at http://www.eclipse.org/

[42] Symbian Software Ltd, Introduction to Carbide.c++, September 2009, available at http://developer.symbian.com/main/downloads/papers/Carbide/Introduction_to_Carbide.pdf

[43] Nokia Corporation, Productions, September 2009, available at http://www.nokia.se/hitta-produkter/produkter

[44] 2N Telecommunications, 2N Netstar-Virtual PBX, September 2009, available at http://www.2n.cz/products/pbx/voip-virtual-system.html

[45] Telia, Telia homepage, September 2009, available at http://www.telia.se/

[46] Wikipedia, Call-through telecom, September 2009, available at http://en.wikipedia.org/wiki/Call-through_telecom

[47] Wikipedia, Call forwarding, September 2009, available at http://en.wikipedia.org/wiki/Call_forwarding

[48] Apache Jakarta Commons FileUpload, September 2009, available at http://commons.apache.org/fileupload/

[49] Wikipedia, Load testing, September 2009, available at http://en.wikipedia.org/wiki/Load_testing

[50] HP Corporate, HP LoadRunner software Data sheet, September 2009, available at https://h10078.www1.hp.com/cda/hpdc/display/main/download_pdf_unprotected.jsp?zn=bto&cp=54_4000_100

[51] Apache JMeter, September 2009, available at http://jakarta.apache.org/jmeter/

[52] Ingenieurbuero David Fischer GmbH, A Guide to Getting Started with Successful Load Testing, 2007, available at http://www.proxy-sniffer.com/en/doc/LoadTestKnowHowEN.pdf

[53] Nokia Forum, Playing audio tones, December 2007, available at http://wiki.forum.nokia.com/index.php/Playing_audio_tones

[54] Shanbo Li, Web Call Example Application, August 2009, Master's thesis, Royal Institute of Technology, August 2009, available at http://shanbohomepage.googlecode.com/hg/master_thesis/master_thesis.pdf

Appendix A Test Results

Table 20: Call Establishment Delay Test Results

Calling Method	Direct Call		Call Back				Call Through			
Application			OptiCaller		Java		OptiCaller		Java	
Mobile Phone	N95	E61	N95	E61	N95	E61	N95	E61	N95	E61
Call Establishment Time (seconds)	7	7	24	17	17	12	18	18	18	20
	6	6	23	16	17	12	19	19	18	20
	7	8	25	18	15	10	19	19	19	19
	6	8	24	16	18	11	16	18	18	21
	8	7	22	16	17	11	17	18	18	21
	8	8	24	17	17	11	20	19	19	22
	7	6	23	18	17	11	18	18	20	20
	7	6	24	16	18	11	19	19	18	21
	6	7	24	16	19	13	17	19	19	19
	7	6	23	16	18	11	18	18	18	21
	8	7	25	19	17	11	19	20	19	22
	7	6	24	15	18	12	19	18	17	21
	7	6	25	16	16	10	18	18	18	21
	7	7	21	16	18	11	20	18	18	20
	7	6	23	16	17	10	19	19	20	19
	6	7	22	16	17	10	19	20	19	20
	5	7	23	16	18	10	18	18	17	22
	6	6	25	16	19	10	20	18	19	21
	6	7	24	16	18	10	18	18	18	21
	7	8	23	17	18	13	18	19	18	19
Average Time	6.8	6.7	23.6	16.5	17.5	11.0	18.5	18.6	19.4	20.5
Variance	0.9	0.5	1.0	0.8	0.8	1.0	0.9	0.5	0.7	0.8

Table 21: Access Point Connecting Delay Test Results

Mobile Phone	Nokia N95	Nokia E61
AP Connecting Delay (seconds)	5	5
	6	5
	5	4
	6	5
	5	6
	5	5
	5	4
	5	5
	7	4
	6	5
	5	4
	6	6
	5	5
	5	4
	5	5
	5	5
	5	4
	7	5
	5	5
	5	6
Average Time	5.4	4.9
Variance	0.7	0.7

Table 22: DTMF tones (10 digits) Transmission Delay Test Results

Mobile Phone	Nokia N95	Nokia E61
AP Connecting Delay (seconds)	6	6
	5	5
	6	5
	5	6
	6	5
	6	5
	5	5
	5	6
	6	5
	6	6
	6	6
	6	6
	6	5
	5	6
	6	6
	6	5
	5	5
	6	6
	6	5
	6	5
Average Time	5.7	5.5
Variance	0.5	0.5

Appendix B
Provisioning System Web Interface

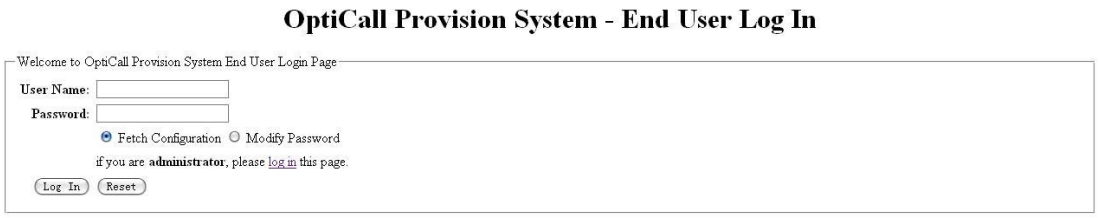

Figure 37: End Users Login Page

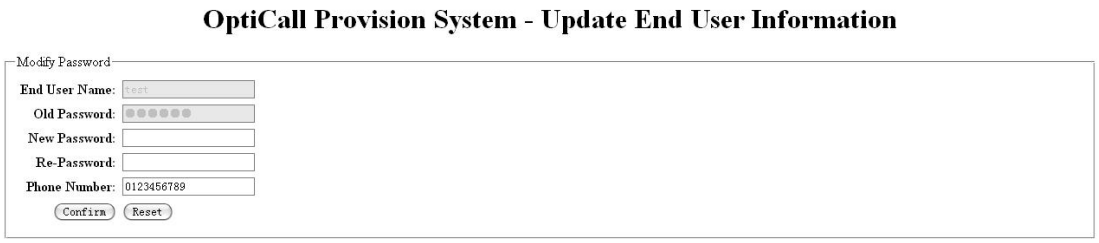

Figure 38: End Users Update Profile Page

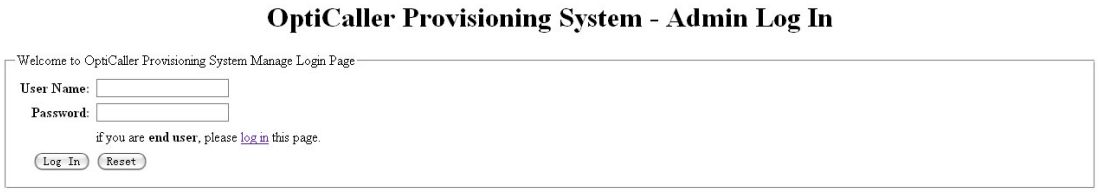

Figure 39: Managers and Manager Login Page

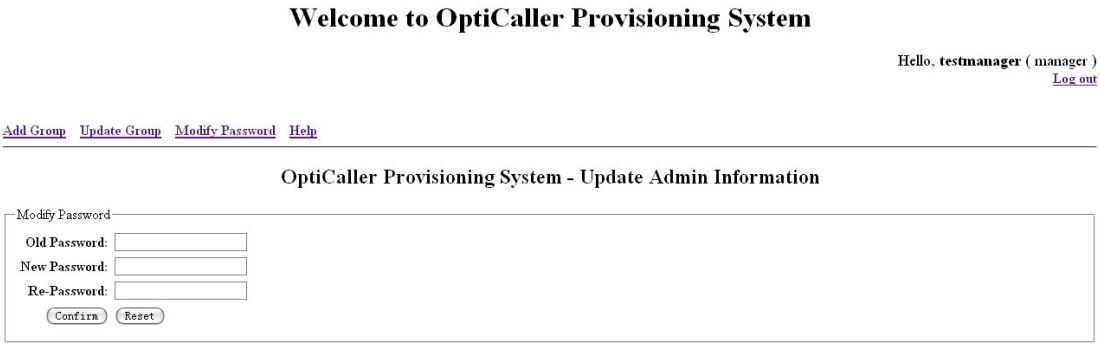

Figure 40: Managers and Administrator Update Profile Page

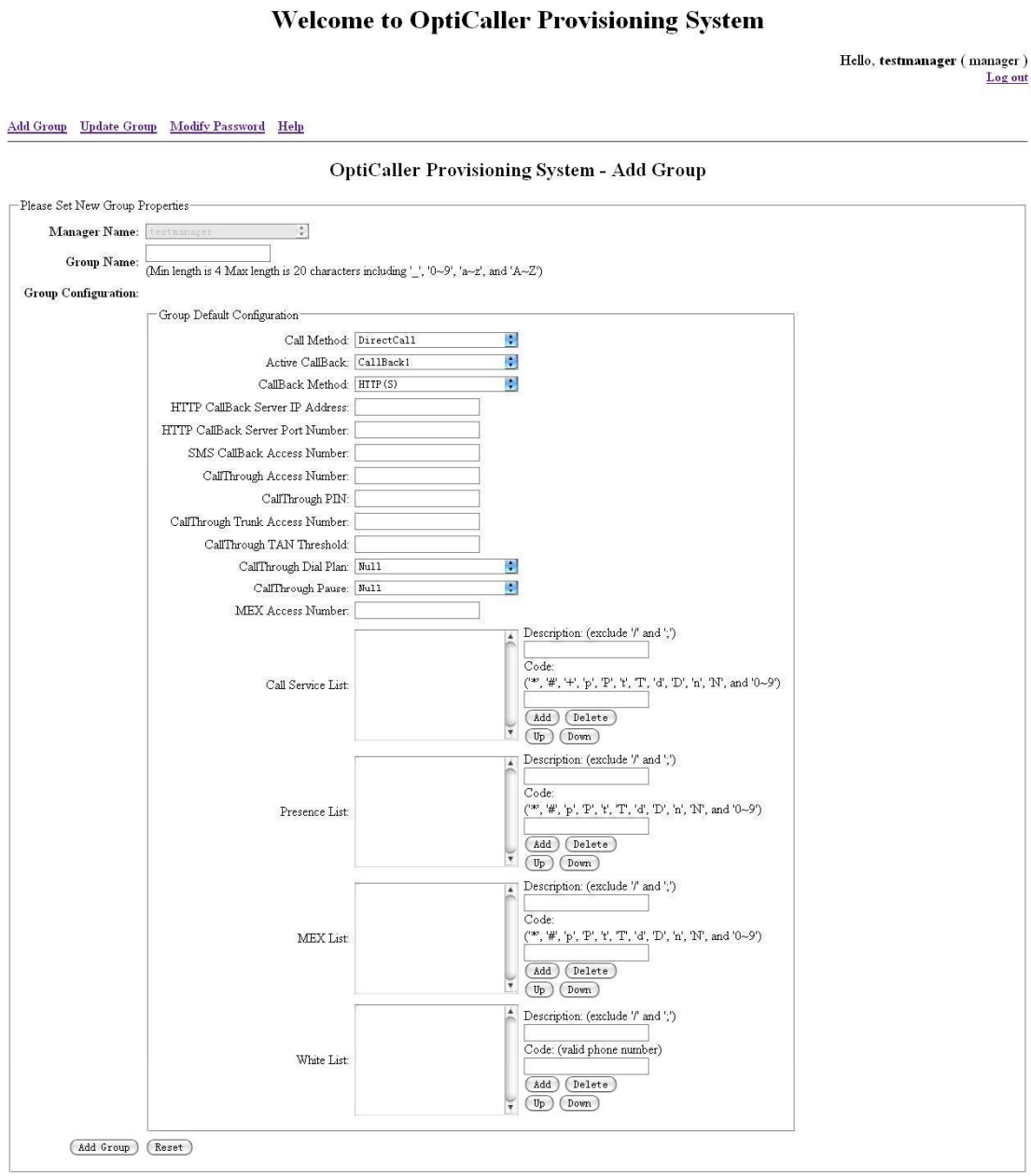

Figure 41: Managers and Administrator Add Group Page

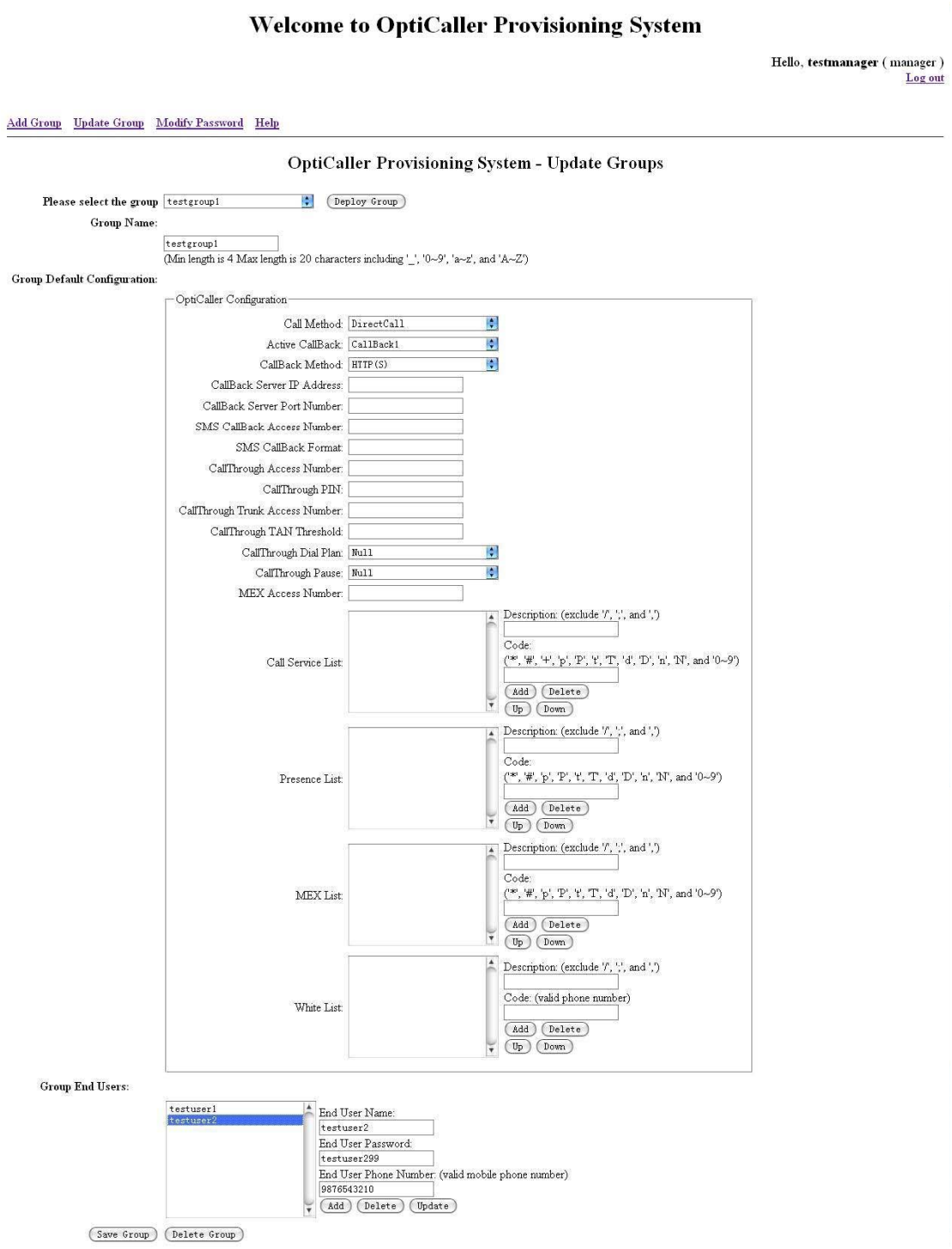

Figure 42: Managers and Administrator Update Group Page

Figure 43: Managers and Administrator Deploy Page

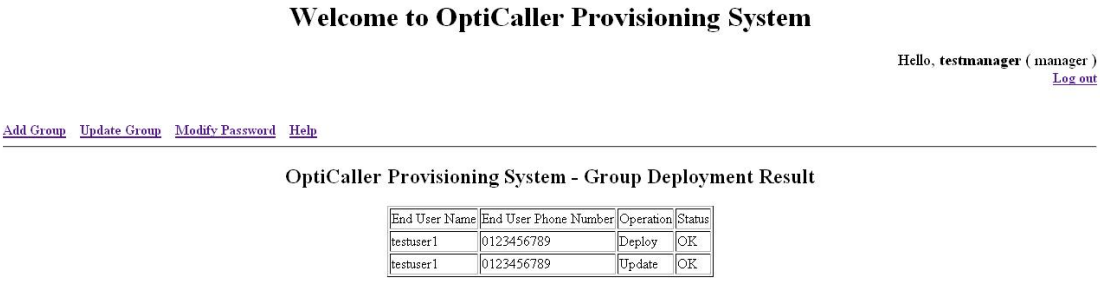

Figure 44: Managers and Administrator Deploy Result Page

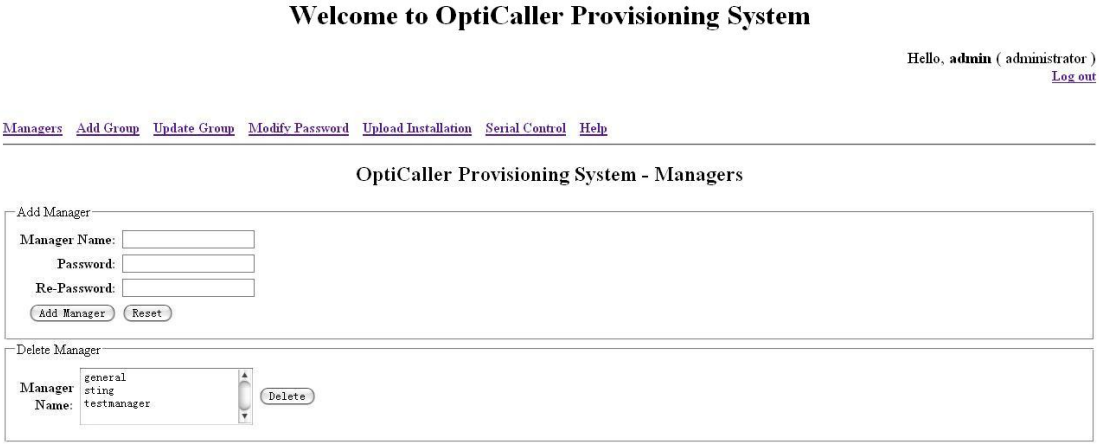

Figure 45: Administrator Manage Manager Page

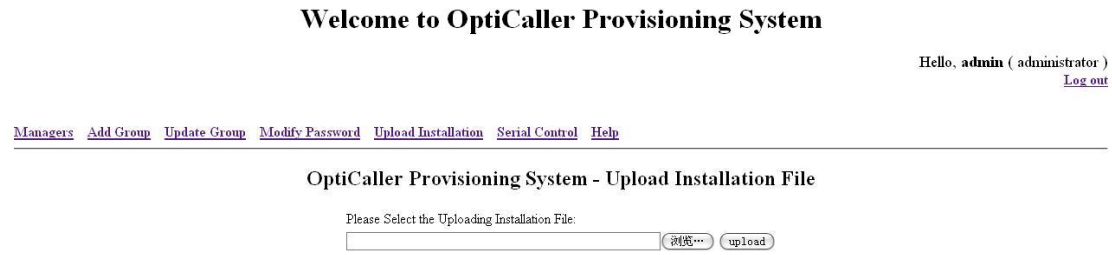

Figure 46: Administrator Upload File Page

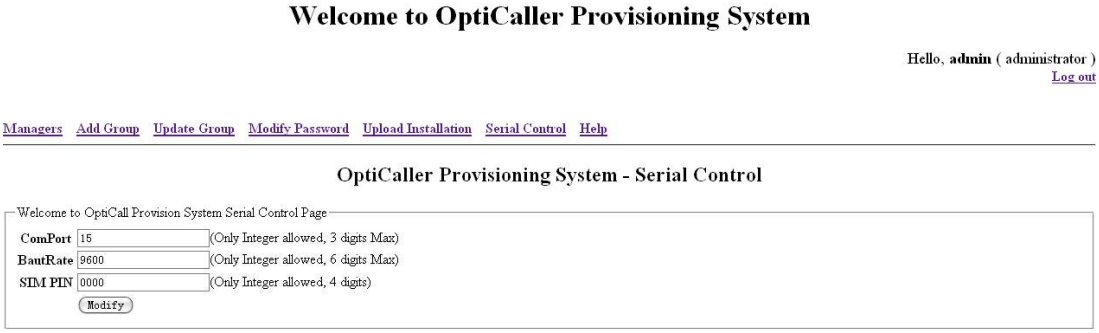

Figure 47: Administrator Serial Control Page

TRITA-ICT-EX-2009:177

www.kth.se