Master Thesis
Software Engineering
Thesis no: MSE-2003-03
March 2003

Designing a Virtual PBX for mobile telephony

- using PARLAY and JAIN technology

David Pettersson

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 10 weeks of full time studies.

Contact Information:
Author:
David Pettersson
E-mail: david@davidgoliat.nu
Phone: +46 703 91 11 99

University advisor:
Håkan Grahn
Department of Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

Parlay is an open standard that focuses on opening up the telecommunication networks for new services and new service providers. It encapsulates the underlying protocols and signaling layers and provides a framework for services so that the applications do not need to handle signaling and network related tasks. Parlay is also created to integrate the public telephone network with wireless and packet based networks. Java APIs for Integrated Networks (JAIN) is the Java implementation of the Parlay specification and includes a Java framework for network access and call control, among others.

I have in this thesis used the Parlay and JAIN technology to design a Virtual Private Branch Exchange (VPBX), which is a software based system for the telecommunications domain. The VPBX provides functionality such as attendant control, routing and re-routing of calls, telephone queue handling and calls on hold. All use of the system is made from mobile phones, even the attendant client application is designed for a handheld device. I have in this thesis presented an architectural design for the VPBX and an evaluation of the design.

Keywords: JAIN, Parlay, PBX, JAIN Service Provider API

CONTENTS

1 INTRODUCTION

The mobile phone industry is getting bigger and bigger. Many people use mobile phones more than regular phones in their work. Most companies have both regular phones and mobile phones but some companies have now changed to use mobile phones only.

Companies with a Private Branch Exchange (PBX) can not use mobile phones in their system the same way as regular phones. It can be a problem that mobile phones cannot be accessed through an extension like other regular phones. This is a problem that gets bigger when the company uses more mobile phones than regular ones.

Ericsson and possibly other companies have provided solutions to connect mobile phones to a company telephone network, the same way as regular phones are connected. But the solution is only for their Enterprise products with lots of extensions and users.

I will focus on the small company that do not need the enterprise solutions, but have employees that are mobile in their work and uses mobile phones more than regular phones. A typical example is a consulting firm that has most of its employees out on the field.

1.1 Project definition

In this master thesis project I will design a Virtual Private Branch Exchange for mobile telephony. This project will result in a design specification that presents a solution for how to implement the system based on the JAIN/Parlay technology and the JAIN framework.

A *Private Branch Exchange (PBX)* is defined by the Dictionary of Computing (1998) as "a small telephone exchange located within a company that allows the people in the company to dial each other or to dial out to an external telephone number". For more information about PBX, see the next chapter.

A *virtual PBX* system is here defined in the sense that a company does not have the physical PBX system. Instead the company subscribes for the telephone exchange service and can access the system via mobile phones and computers. The physical system is located either at the network operator or by a third party service provider. This way the company does not need to invest in expensive equipment.

I define a *mobile PBX* system in the sense that the use of the system is done with wireless phones and such appliances. In this case I define mobile as the use of mobile phones (cellular phones) in the GSM and/or future 3G networks. Due to the possibilities with JAIN technology, any telephone like appliance that use either PSTN, packet based or wireless networks can in theory be used. In this thesis I will only focus on the use of mobile phones, for both calls and administrative purpose.

1.2 Problem statements

I want the following problem statements answered:

- How to design a virtual PBX system for mobile telephony?
- Can it be done with the Parlay and JAIN technology?
- Is Parlay and JAIN a good technology for this type of product?

The project will produce an architectural design specification. This specification will be evaluated against the functional requirements. A discussion on some quality requirements will also be presented.

1.3 Context

The technology that I focus on is the Parlay and JAIN technology. Parlay is an open standard developed by many of the leading companies and institutions in the telecom industry. Parlay focuses on opening up the telecom networks for new services and new service providers. It also encapsulates the underlying protocols and signalling technology and provides a framework for services so that the applications do not need to handle signalling and network related tasks in any bigger extent. Parlay is also created to integrate the public telephone network (PSTN) with wireless networks and packet based networks like IP and ATM networks.

JAIN stands for Java APIs for Integrated Networks and is developed by Sun Microsystems in cooperation with the Parlay group and leading companies in the telecom domain. JAIN is the Java implementation of the Parlay specification and includes a Java framework for network access and call control, among others.

As this thesis is limited by its size in number of hours (400h) I had to constrain the size of the thesis. My intention was to create a prototype to test the design proposal that is the result of this thesis. Due to the time constraints this has not been possible.

2 BACKGROUND

2.1 Public Switched Telephone Network (PSTN)

The oldest and largest telecommunications network today is the public switched telephone network (PSTN) which has more than 700 million subscribers worldwide.

The PSTN has been in use since the late 19th century, and has gone through many changes and improvements since then. Most changes to the PSTN have been made since the 1960s. Some of these innovations include: data communication, telefax, digital voice transmission, satellite communication, digital switching, optoelectronics, network intelligence structures and the Internet (Olsson et al. 1998).

For a long time, the PSTN was the only bearer network available for telephony. Today, many people choose the mobile telephone for their calls. Other bearer networks for voice transmission include integrated service digital network (ISDN), asynchronous transfer mode (ATM), frame relay and the Internet (Olsson et al. 1998).

The PSTN has analog access; therefore most of the telephones used today are still analog, and for data communication; modems that convert the digital data to analog signals are used. The PSTN is circuit-switched and also duplex, which means that data can be sent in both directions. A reference model of the PSTN can be seen in figure 2.1.

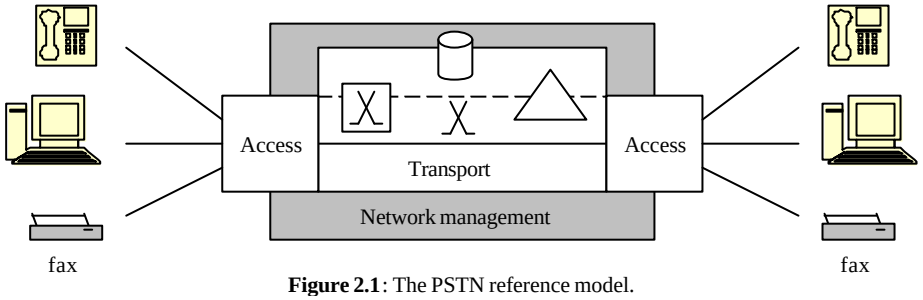

Figure 2.1: The PSTN reference model.
(Sourced from Olsson et al. 1998)

2.2 Integrated Service Digital Network (ISDN)

The traditional network layout has one network for telephony, a second for telex, a third for packet-switched data traffic, and so on. This is of course not an ideal situation from a network operator's point of view, considering today's cost of network management. The transition from analog to digital technology forced the network operators to integrate digital switching and digital transmission into their telecom networks (Olsson et al. 1998).

The integrated services digital network (ISDN) was developed to meet this new demand. ISDN is now integrated into the PSTN and enables the subscriber to access the telecom network through a digital connection. ISDN enables both voice and data communication. ISDN also provides higher bandwidth (mostly 64 or 128 kb/s) for data traffic than the PSTN (with modems that use 56 kb/s).

2.3 Private Branch Exchange (PBX)

Many companies have a large number of telephone lines and data connections. Not all telephones need an outgoing line to the telecommunication network at the same time. Therefore a Private Branch Exchange (PBX) is a good solution for larger companies. All telephones inside the company connect to a PBX This PBX then

connect to the PSTN, often with a digital ISDN connection. Calling within the company is free since the PBX only connects to the PSTN for outgoing calls. The PBX works like an interface between the local network and the public telephone network.

A PBX can provide a large range of functions and services (Olsson et al. 1998). Some of them are listed here:

- Telephony systems which enables direct dialling-in
- Key systems, in which some of the connections serve as telephonist positions and receive and transfer calls
- Automatic call distribution
- Handling of voice and text messages
- Call centre
- Interactive voice response
- Cordless extensions
- ISDN extensions able to transmit voice, data and images via the PBX

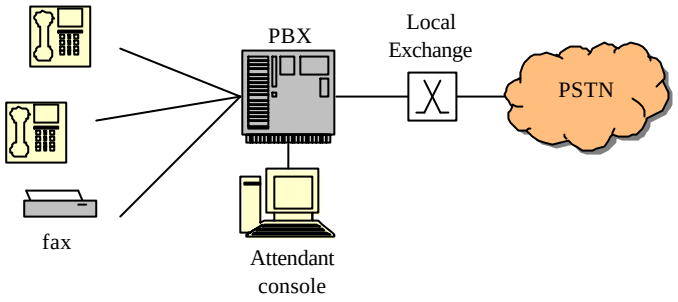

Figure 2.2: Overview of a Private Branch Exchange (PBX)

PBXs have been in use since the 1960s, and the technology has evolved through four generations. From very easy analog systems, to today's advanced digital enterprise solutions that connects thousands of telephones over many countries to one local telephone network. PBXs have been proprietary in everything from telephone instruments to operating systems, but have now been challenged by new systems that run in a PC with a commercial operating system like Unix or Microsoft Windows (Green, 2000).

A function that is very important for PBX solutions are the possibility to have attendants (telephonists) that receive incoming calls and redirect them to the destination. Most PBXs have attendant consoles for both incoming calls and supervision. Another function that is often used is the telephone queue, which puts incoming calls on a queue if the attendant is busy.

2.4 Public Land Mobile Network (PLMN)

Mobile telephony is one of the fastest growing and most popular teleservices that has ever existed. The public land mobile network (PLMN) is a telecommunications network for mobile units, mostly referred to as mobile stations, mobile phones or cell phones.

The technology for mobile telephony has progressed from analog systems (ex: NMT) to digital (ex: GSM) and now soon third generation systems (ex: 3G). Today GSM is the most used system for mobile telephony and I will therefore describe the GSM a little bit further here.

The network structure of the PLMN is much more complex than the PSTN. The access to the network is radio based instead of wire like the PSTN. The mobility

requires that the network always can detect the phone so that calls always can be carried out. The system must also be flexible so that the user can move around and still be able to uphold the call. Some of the functions the network must handle include location updating, roaming, handover and paging (Olsson et al. 1998).

In figure 2.3 you can see the network structure of the GSM network. The base station controller (BSC) includes a switching function that allows it to switch to another base station (BTS) when the terminal (MS) moves. This is called roaming. If the terminal moves to a base station that the BSC does not control, the mobile switching centre (MSC) handles the transfer to the new BSC that controls that base station. The mobile network also needs functions for network intelligence. The figure shows two of these functions: the home location register (HLR) and the visitor location register (VLR).

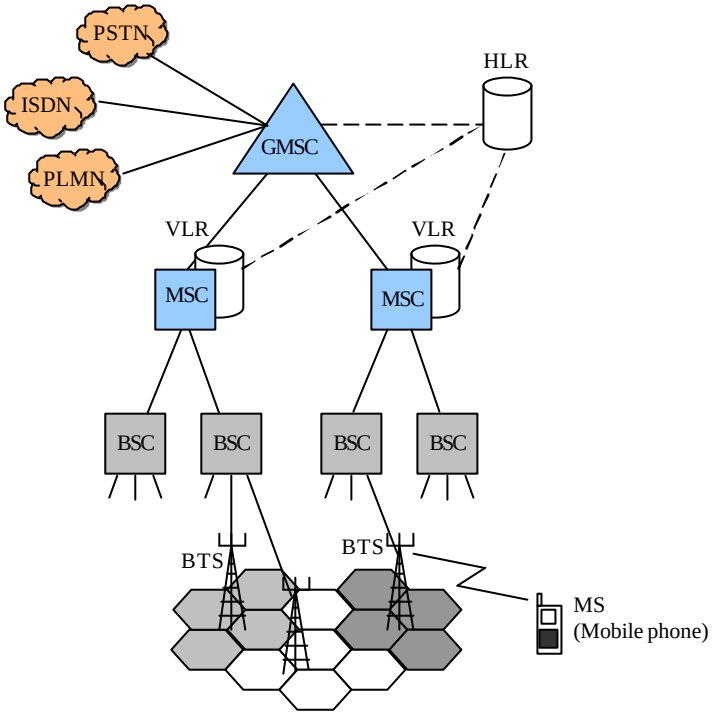

Figure 2.3: Overview of the GSM network with the main network elements.

Each base station controls one or more cells. Each cell has a number of radio frequencies that are used for communication with the terminals.

The main network elements used in the GSM networks are:

- MS: *A mobile station* can be many different devices. One is a mobile telephone; another is a laptop computer with a radio modem.
- BTS: A *base transceiver station* handles transmission and reception. It has antennas for one or more cells. It also has equipment for signal strength measurements and communication with the BSC.
- BSC: A *base station controller* sets up the radio channels for traffic and signalling to the MSC. It also monitors the access network portion of the connection. A BSC also handles handover between the base stations that it controls.
- MSC: A *mobile switching centre* is a specialised switching node. The MSC can be linked to other PLMNs and to other MSCs. A PLMN can have one or several MSCs, depending on the size of the network and the number of

subscribers. The MSC handles handover between different BSCs, MSCs and other PLMNs. The cells that a MSC controls are called the MSC service area.

- GMSC: A *gateway MSC* is a specialised MSC that serves as an interface to other networks. All connections to and from mobile networks pass through a GMSC. The GMSC handles subscriber data and charging of accounts between different networks. The GMSC has connections to PSTN and PLMN networks at both the national and international level.
- HLR: A *home location register* is a database for permanent storage of subscriber data. In a fixed network, every subscriber belongs to a local exchange. In a mobile network the subscriber instead belongs to the network, and the HLR is therefore used to store information related to the subscriber. It keeps track of the location of the subscriber, whether he is in an MSC service area or in a different PLMN. This information is used by the GMSC to locate the user, when it receives a call from another network.
- VLR: A *visitor location register* is used to store information about the temporary subscribers in the network, namely those who happen to be in the MSC service area at any given time. The VLR keeps track of in which service area cell the mobile station is located, and is informed if the mobile is turned ON or OFF.

2.5 Parlay

In 1998, BT, Microsoft, Nortel, Siemens, and Ulticom joined forces in the Parlay Group in order to create a framework or a set of APIs to enable applications to run outside the secure telecom network domain and still have access to all the resources inside the telecom network. Since then the Parlay Group has increased with many new members. Some of them are: AT&T, Cegetel, Cisco, Ericsson, IBM and Lucent.

The Parlay Group has according to Beddus et al. (2000) developed a secure API that offers a wide range of features while providing scalability and extensibility. The Parlay API enables network operators, service providers, and the general software creation and sales population to integrate telecommunications capabilities into generic IT software, taking advantage of information that is both private to the end user and real-time in nature.

The goal of the Parlay Group is to open up telecommunications capabilities in the same manner in which the personal computer (PC) was opened up, so that software developers can be more creative and innovate, when developing new services for the telecom domain.

The Parlay API defines a set of technology independent interfaces that specify methods, events, parameters, and their semantics to allow external (untrusted third parties) and internal (traditional network operators) application creators the control over core network resources and capabilities. The Parlay API has been translated into popular technology definitions that for example cover Microsoft's Distributed Component Object Model (DCOM) and the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA).

The Parlay API describes the interaction between two computers, one in the secure network domain and one outside in the enterprise domain, or perhaps in the palm of your hand. The application executes outside the secure network domain, but uses the network capabilities offered inside the secure network, through the Parlay API.

According to Beddus et al. (2000), the focus of the Parlay Group has been to define a technology independent API for controlling voice and data networks. To make the Parlay API useful to the real world, distributed computing technology such as DCOM and CORBA has been specified for the protocols between the client machine

and the network API server. Beddus et al. (2000) explains this approach as being very flexible since it is not tied to any specific programming language: C, C++, Java and Visual Basic, among others, can all be used. The drawback, according to Beddus et al.(2000), is that the developer must be skilled in three areas: the Parlay API distributed computing techniques, and programming languages.

Parallel to the Parlay Group, another group was working with the same idea and goal. This was a group that was working with a similar API for the Java domain, called Java APIs for Integrated Networks (JAIN). In July 1999 a cooperation between the Parlay Group and the JAIN Community was formed called The JAIN Parlay Edit Group. The new group started working with the Parlay API and has created a Java version of the Parlay API, which of course is called: JAIN.

The JAIN Service Provider API (JAIN SPA) is the framework the application programmers use to communicate with the parlay server inside the secure network domain. The JAIN SPA is based on the Parlay specification and is the Java implementation of the Parlay API (Jepsen et al. 2001).

Since JAIN is an implementation of Parlay and the JAIN framework is available for application programmers today, I will focus on JAIN throughout the rest of this report. The next chapter will explain JAIN (and indirectly Parlay) in much more detail.

3 JAVA APIS FOR INTEGRATED NETWORKS

3.1 What is JAIN?

The Java APIs for Integrated Networks, JAIN, is a set of APIs for the Java platform, which provides a framework to build and integrate services that span across packet, wireless and PSTN networks. This enables voice, data and multimedia communication over Internet, GSM/3G wireless networks and the public telephony network simultaneously.

JAIN technology builds on Java portability by standardizing the signaling layer of the communications network into the Java language, and defines a communications framework for services to be created, tested and deployed. The strengths of JAIN are in service portability, network convergence, and secure network access (Jepsen et. al. 2001, Keijzer et al. 2000).

- ***Service portability*:** Technology and application development are currently constrained by proprietary interfaces. Almost no portability of applications exists. This increases application development cost, time to market, and maintenance requirements. The JAIN approach is to create uniform Java interfaces that can deliver portable applications that are not based on proprietary interfaces.
- ***Network convergence*:** Call or session legs for most of today's applications and services typically span only a single type of network – public switched telephony network (PSTN), packet, or wireless – although clearly gateways between these networks do exist. The higher-level JAIN call models include facilities for observing, initiating, answering, processing, and manipulating calls, where a call is understood to include a multimedia, multiparty, multi-protocol session over the underlying integrated network.
- ***Secure network access*:** Normally communication applications and services run either inside the operator's trusted network or completely outside this network. The JAIN Service Provider API interface makes it possible for untrusted services, residing outside the network, to directly access network functions inside the integrated network. This makes third party development of telecommunication services possible in a secure manner.

According to Jepsen et al. (2001: 17) JAIN technology enhances the convergence of the Internet and PSTN by providing controlled access of untrusted services to available functionality and intelligence inside the operators' networks. Using the Parlay and JAIN Service Provider API (JAIN SPA) makes services become network-operator-independent. This will attract service providers and third parties to develop interesting services for the telecommunications market. The focus of the JAIN effort is therefore to take the telecommunications market from today's many proprietary systems to a single open, distributed environment, not unlike the Internet today. This is now possible with Java technology using the JAIN APIs.

3.1.1 JAIN Community Process

The JAIN API is defined and specified by the JAIN Community, which is a large number of communication companies what work together to create the JAIN open specification. The JAIN Community is working according to the Java Community Process, JCP, which is a formal process for developing Java extensions. According to Keijzer et al. (2000) the JCP produce high-quality specifications that not only deliver the specification, but also the reference implementation and its associated suite of compatibility tests.

The JAIN standardization effort is organized in two broad areas: the Protocol Expert Group, PEG, and the Application Expert Group, AEG. The Protocol Expert

Group standardizes interfaces to IP, wireline, and wireless signalling protocols. These protocols include for example SIP, MGCP, H.323, TCAP, ISUP, INAP/AIN and MAP. The Application Expert Group deals with the APIs required for service creation within a Java framework and presents specifications for secure network access, connectivity management, JAIN Call Control, JCC, and carrier grade service logic execution environment, JSC/SLEE.

JCP also consist of work groups, which develop prototype implementations of sub specifications and feeding the results back into the Expert Groups. Examples of workgroups are AT&T, KPN, CMG and Ericsson.

3.1.2 Reference implementation and commercial products

The reference implementations and compatibility tests from the JCP are only made to verify that the specifications work and delivers the high quality that is specified. No full implementation of the framework are delivered by either the Parlay Group or the JAIN Community. They only deliver the specifications. Therefore full implementations on the framework are left open for comercial companies to develop in their own way based on the specifications.

Applications that use the functions in the framework are then developed towards these commercial products. There are today only a few comercial products on the market that are built on the JAIN specification and therefore the amount of applications and products that use the JAIN framework are very limited. For more up to date information about commercial products see the JAIN and parlay webpages (for URLs see Internet links at the end of this report).

3.1.3 JAIN architecture

As said before, JAIN integrates wireline, wireless and packet-based networks by separating service-based logic from network-based logic. While doing this JAIN will achieve service portability, network convergence and secure network access.

Based on the JAIN layered approach illustrated in figure 3.1, the JAIN architecture consists of two layers, application and protocol.

The protocol layer standardizes interfaces to IP, wireline, and wireless signaling protocols for the Java language. Some of the protocols are: TCAP, ISUP, INAP, MAP, MGCP, Megaco, and H.323. The protocol interfaces are made in a Java object model, which creates an architecture that enables protocol stacks to be dynamically interchanged, and offers portability to the applications since protocol stacks can come from different vendors.

The application layer in the JAIN architecture provides a single call, or session, model across all supported protocols in the protocol layer. The fundamental idea is to provide a single interface for the applications or service components to access the networks. The application layer defines service provider APIs (JAIN SPA), Java Call Control (JCC), JAIN service creation environments (JAIN SCE), and carrier grade JAIN service logic execution environment (JAIN SLEE), which will be described later in this chapter.

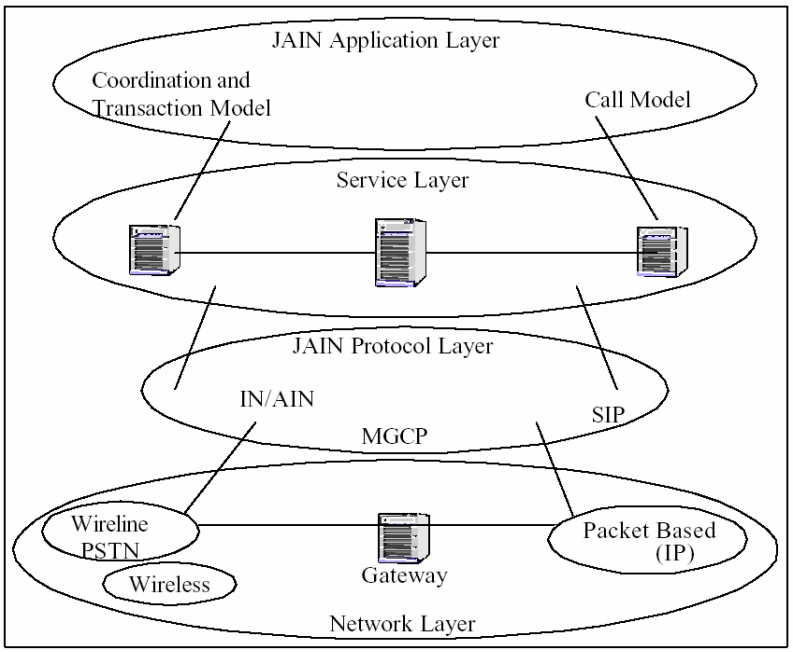

Figure 3.1: The layered approach for JAIN.
Sourced from (Sun Microsystems, 2002).

The JAIN architecture supports three abstractions, which are illustrated in figure 3.2. The lowest level of JAIN abstractions is in the protocol layer, where applications or services can talk directly to the JAIN adapters. These adapters are Java class methods, call backs, events, or Java interfaces that encapsulate the underlying resources. The resources can be implemented in any language such as Java, C, C++ or others, but should at least provide a relevant JAIN Java API. This level of abstraction does not provide any features to the application for dealing with different kinds of protocols, but it does provide for applications to run on top of protocol products from different vendors. As explained by Jepsen et al. (2001), an application that needs a session spanning INAP and SIP will have to handle both protocols, but those protocol products can come from different vendors.

The next level of JAIN abstraction is the Java Call Control level. An application at this level does not need to know that some of its sessions or call legs are using different protocols. The JCC provides a very useful interface for applications to create and/or handle calls, where a call refers to a multimedia, multiparty, multi-protocol communications session (Jain et al. 2000).

A third level of abstraction is provided with JAIN SPA. In figure 3.2 the S outside the SLEE box represents JAIN SPA based services. These are untrusted services that can access the network through the JAIN SPA interface, or JAIN Parlay as it is also called. The JAIN SPA acts like a firewall to protect the security and integrity of the integrated network. According to Keijzer et al (2000) some operators might opt to have all services both inside and outside their integrated network domain, use the JAIN SPA interface. The JAIN SPA addresses some or all the capabilities of the underlying abstraction levels. Examples of these capabilities are: multiparty call control, messaging (i.e. short messaging service, SMS), user location, user status, and user interaction.

The vertical bar labelled OA&M in figure 3.2 is a set of APIs for Operational, Administrative, and Maintenance aspects of the JAIN environment.

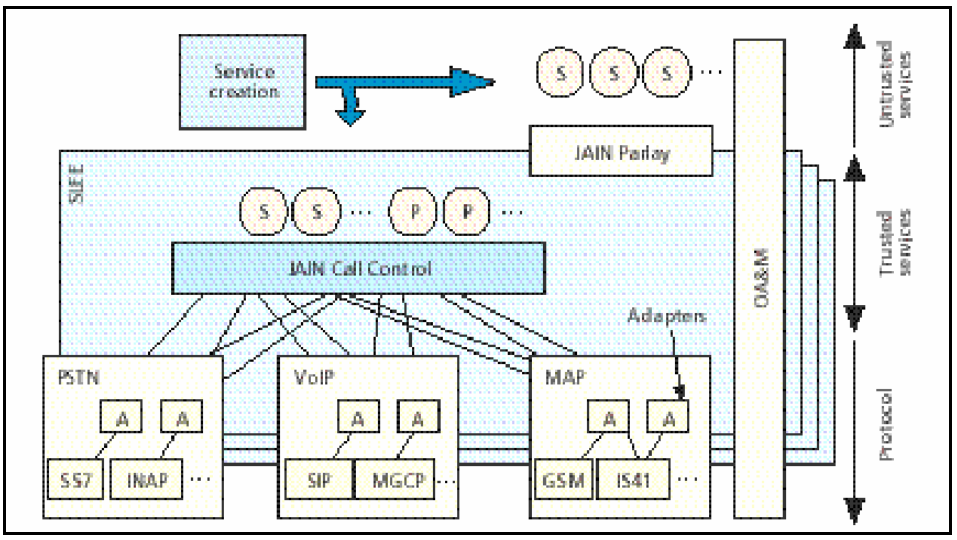

Figure 3.2: JAIN abstractions.
Sourced from (Sun Microsystems, 2002)

The JAIN SLEE box in figure 3.2 is an execution environment for JAIN services. JSLEE stands for JAIN Service Logic Execution Environment. It focuses on a component model over specific implementations. According to Keijzer et al. (2000) the Enterprise JavaBeans (EJB) is an excellent candidate for such a component model, but other component models such as Jini and others can also be used.

JSLEE also provides other features such as: portable support for transactions, persistence, load balancing, security, object and connection instance pooling, and so on (Keijzer et al. 2000).

3.2 JAIN Protocol API

3.2.1 JAIN SS7 API

The JAIN SS7 API defines Java APIs for the following protocols:

- **TCAP**

Transaction Capability Application Part (TCAP) is a transaction-based protocol and creates a layer in the SS7 protocol. It is used to transfer information from one application at a switch to another application within another network entity. The communication between the entities is carried out with a series of message exchanges, or dialogs. Components are the specific operation requested by the application and the dialogs are carried out as transactions.

The JAIN TCAP API specifies the Java interfaces and classes that are required to interact with the TCAP components. The API includes features for initialising and terminating a TCAP transaction, managing TCAP dialogue identifiers, and building and sending TCAP transactions and components (Jepsen et al. 2001).

An example of an application in the real world that uses TCAP is the “800 Number Translation service”, which is used to convert a virtual 800-phone number to a physical routable phone number (Sun Microsystems, 2002).

- **ISUP**

Integrated Services Digital Network User Part (ISUP) is a call-related signalling protocol used within the SS7 protocol. It is used for call set-up, and to manage and

release trunk circuits that are used to carry voice and data calls over the public telephone network.

The JAIN ISUP API provides the necessary Java API to the signalling functions in the telephone network, which is used to create voice and data applications (Sun Microsystems, 2002).

- **MAP**

Mobile Application Part (MAP) is a non-call-related signalling protocol that is used for interactive mobile applications in the cellular GSM network.

The JAIN MAP API provides interfaces for the mobile network. JAIN MAP is concerned with features such as Short Messaging Service, user location among others (Sun Microsystems, 2002)

- **INAP**

Intelligent Network Application Part (INAP) is a non-call-related control protocol that allows applications to communicate between various nodes of an Intelligent Network. The JAIN INAP API defines the operations to be performed between service providers for providing IN services, such as number translation, time of day, follow me etc.

3.2.2 The JAIN IP APIs:

Telephone services have up till the wide spread of the Internet been carried out on traditional circuit switched networks. Now packed based networks such as the Internet have evolved and inspired industry pioneers to create services and technology for traditional voice and multimedia traffic over the Internet. To enable services over an IP-network equivalent to those of a traditional telephone network, various new protocols, such as H.323, MGCP (Media Gateway Control Protocol), SIP (Session Initiation Protocol), and various new network entities, such as media gateways, gatekeepers, call agents and media gateway controllers have been defined.

Multimedia terminal: The multimedia terminal is directly connected to the IP network. The terminal could be a PC with special software, or can be a dedicated appliance such as an IP phone. The protocols used for setting up multimedia calls can be H.323, SIP or MGCP.

Media Gateway (MG): The MG is used to connect the IP telephony network with the existing PSTN. It works like a gateway between the two networks. The MG is a relatively simple device that translates circuit-switch voice to packet voice.

Media Gateway Controller (MGC) / call agent (CA): The MGC controls the MG through MGCP to set up connections between the PSTN and the IP network. The MGC represents the call signalling intelligence in the IP network.

Gatekeeper (GK): The GK is an H.323 control entity the controls access to back-end services, such as directory lookups for call routing among others. The GK may be implemented as a standalone entity or combined with the MGC.

There are a wide variety of devices and platforms that can be used to provide multimedia services over IP networks. The JAIN IP API defines Java-based APIs for the following protocols.

- **H.323**

H.323 provides support for the transmission of real-time audio, video, and data communications over packet-based networks, such as the Internet Protocol. H.323 specifies the components, protocols, and procedures for multimedia communication over packet-based networks. The H.323 protocol enables multimedia communication with both audio and video. The JAIN H.323 API is a standard, generic Java-based interface to native H.323 protocol stacks. (Jepsen et. al, 2000).

- **SIP**

  Session Initiation Protocol (SIP) is a signalling protocol for handling sessions with one or more participants. These sessions include Internet multimedia conferences, Internet telephone calls or voice over IP (VoIP), and multimedia distribution (Jepsen et. al, 2001). Like HTTP protocols SIP is an application client/server protocol, were SIP clients issue requests and SIP servers responds to the request. SIP does not include the mechanism for media streaming between caller and "calee" (Sun Microsystems, 2002). The JAIN SIP API provides an industry standard interface into proprietary SIP protocol stacks. It is a standard, generic Java-based interface that sends and receives SIP control messages to and from the native SIP protocol stacks in SIP clients and SIP servers.

- **MGCP**

  Media Gateway Controller Protocol (MGCP) controls voice and media over packet gateways. As described before the media gateways connect PSTN with packet networks. This allows users to make calls that span both the PSTN and packet networks. MGCP has been defined as a protocol for controlling these gateways.

  The JAIN MGCP API allows developers to write MGCP services while the standard is maturing. The JAIN MGCP API will be backward compatible as the MGCP protocol is enhanced. JAIN MGCP API provides an industry standard Java technology interface into proprietary MGCP protocol stacks. The API includes gateway interfaces and control interfaces necessary to create and release connections, to modify connections, and to audit connections (Sun Microsystems, 2002).

3.3 Java Call Control API

3.3.1 Java Call Control

The Java Call Control API (JCC) is created to define a programming interface to next-generation converged networks (PSTN, packet based and wireless). It is designed to hide the details of the underlying network and protocols for the application programmer, in order to create an easier programming environment.

JCC is a call model that handles call related tasks such as creating, manipulating, controlling and tearing down calls. The word "call" is traditionally defined as a two-party point-to-point voice call. In JAIN and JCC a "call" instead refers to a multimedia, multi-party, multi-protocol communications session over the underlying integrated (IP, ATM, PSTN, wireless) network (Jepsen et al. 2001). "Multi-protocol" is described by Jepsen et al. (2001), in the sense that different legs of the call, represents the logical connection to individual parties of the call, may be affected by different underlying communications protocols over different types of networks. He gives one example where one leg of the call may be affected using the H.323 protocol, another via SIP, and a third via traditional PSTN signalling protocols like ISUP.

3.3.2 Call models

Several call models and APIs for call manipulation have been developed in the past. The most important call models are; Advanced Intelligent Network (AIN), Java telephony API (JTAPI), and Telephony API (TAPI). There are many differences between these call models, but they all have a common goal; to initiate, control and manipulate calls.

The AIN call model were designed for the PSTN and provides architecture were telephone switches perform the basic call processing functions. It introduced the concept of triggers that is small software components that executes if certain criterion is met. Useful functions like toll-free-numbers have been created using this technique.

JTAPI and TAPI are made for another domain. Their focus is call processing and applications for a private branch exchange (PBX) or call center environment. TAPI and JTAPI provide a simple object-oriented framework for call manipulation, which enables faster and easier development of applications.

The JCC call model is created with the knowledge gained from existing models like AIN and JTAPI and it captures the essential aspects of these call models. Jain et al. (2000) states that the JCC API provides the application programmer with a convenient and powerful abstraction for manipulation calls and managing the interaction between the application and calls.

Some of the techniques and architecture of JTAPI has been reused in JCC. JCC has also integrated the trigger approach from AIN, as well as event filters and listeners. Apart from this a lot of new features and techniques has been developed to make JCC a flexible and useful call model.

3.3.3 Structure of the API

The Java Call Control API is structured into the following three functional areas:

- **Java Call Processing (JCP)**
  The JCP package includes the basic features for monitoring calls. Tait (2001) states that JCP is too elementary for many if not most carrier-grade deployments. But he continues by saying that JCP is a cornerstone, developed to unify the call control APIs developed by the JTAPI, JAIN and Parlay expert groups. Therefore it only has the basic components for manipulating call.

- **Java Call Control (JCC)**
  The JCC is derived from JCP and inherits all functions from the JCP. The JCC package includes the facilities required for observing, initiating, answering, processing and manipulating calls, as well as invoking applications and returning results during call processing. It is a more complete package that provides the necessary functions for most types of applications. It is the JCC that will be mostly used in development of applications based on the JAIN framework (Jepsen et al. 2001).

- **Java Coordination And Transactions (JCAT)**
  The JCAT includes the same functionality as JCC, but extends JCC in the area of call control, and enables all common AIN applications as well as other integrated voice/data and next-generation services.

3.3.4 Components of the API

The JCP and JCC define four key objects that are common for both APIs. How these objects relate to each other is shown in figure 3.3. The objects are (Jepsen et al. 2001):

- **Provider:** A Provider is the entity that a call control application has to access in order to initiate a call. An application does not create a Provider object directly but can access it through the API. Jepsen et al. (2001) defines the Provider object as the "window" through which an application views the call processing.
- **Call:** The call object represents the call and brings two or more connections together. It is the main object from which connections are created.
- **Address:** The address object represents a logical endpoint, such as a telephone number or IP address.

- **Connection:** The connection object handles connection related tasks such as setting up and tearing down a connection. It represents the dynamic relationship between a Call and an Address.

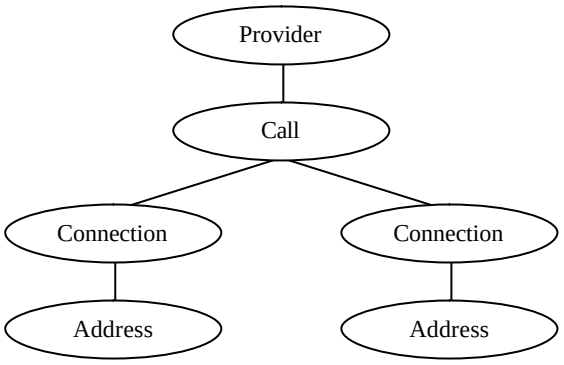

Figure 3.3: Object model of a two party call.
To create a three party call; a new pair of connection and address objects can just be added to the Call object.
Sourced from (Jepsen et al. 2001)

Besides these four key objects, the API resides on events and listeners. The platform or network element implementing the API can inform the application of underlying events, such as incoming calls, by means of Java events. The application provides Listener objects corresponding to the event that it is obtaining.

Since several objects in the API can generate events, which in turn can be trapped by different Listeners written by the application programmer, the Event and Listener objects are organized by inheritance. A diagram describing the inheritance is shown in figure 3.4. The Provider event indicates any state change in the Provider object. The same goes for the other events; the Call event indicates any state change in the Call object, and the Connection event indicates any state change in the Connection object. These events are then reported to their respective listeners, Provider event to the Provider listener and so on.

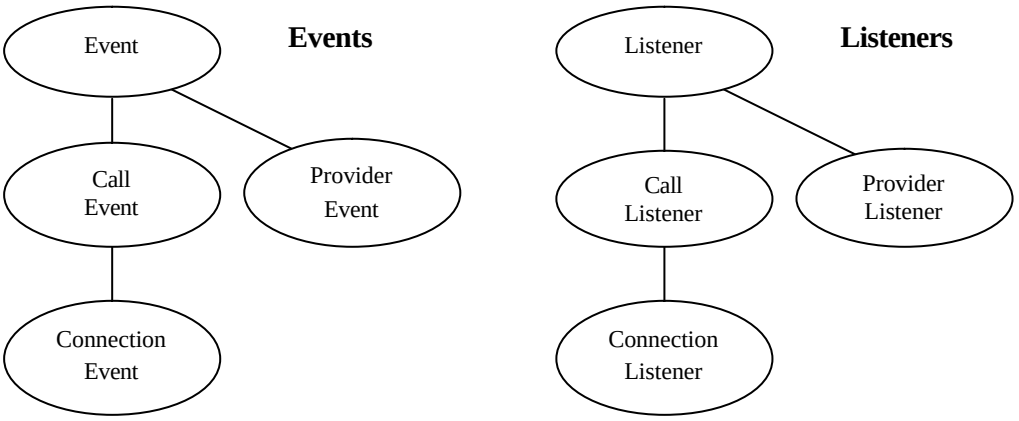

Figure 3.4: JCC and JCP Event and Listener inheritance diagrams.
Sourced from (Jepsen et al. 2001).

3.3.5 Java Call Control (JCC)

Since JCC is a more complete API than JCP, and also inherits most features of JCP, I will here focus only on the JCC API.

JCC has the four key objects described above; Provider, Call, Connection, and Address, with the prefix Jcc on the objects, for example; JccProvider, JccCall etc. All

these Jcc objects inherit from their Jcp counterparts, JccProvider from JcpProvider etc. According to Jepsen et al. (2001) the JAIN specification of the JCC explains the four key objects as I present them here. This is just a brief presentation of the objects. For more detailed information I refer to the JAIN JCC specification.

JccProvider

The JccProvider is used by call control application in order to initiate a call. It is not created directly by the application, instead the application get access to the JccProvider using the `getProvider()` method. The JccProvider is used to create JccCall objects, EventFilters and Listeners, and to add Listeners to different call control objects in order to detect state changes in these objects.

JccCall

The JccCall is an object that represents a call between two or more parties. It is created by invoking the `createCall`() method in the JccProvider interface. For outgoing calls, the application needs to create the JccCall object, but for incoming calls the platform will create it. Other important methods in JccCall are; `createConnection`(), `routeCall`(), and `release`().

The `createConnection`() method just creates a new JccConnection object. The `routeCall`() method creates a JccConnection object and routes this call leg to the signalling network. It results in the call having been successfully routed to one endpoint. `routeCall`() needs to be called for every Connection or call leg to be routed. The `release`() method is used to release a call and thereby freeing up resources used in the network.

JccConnection

The JccConnection object handles the communication with the signalling network. It has a more complex finite state machine than the other key objects. The `routeConnection`() method communicates with the platform and sends messages to the signalling network to route the connection to an end point. The JccConnection object uses the information stored in the JccAddress object when routing the call leg or connection.

JccAddress

The JccAddress represents an endpoint in a communication. The endpoint can be either a telephone number in PSTN or wireless network, or an IP address. Each address can be part of more than one call, and in each case, is represented by a separate Connection object.

4 DESIGN PROJECT AND USE CASES

4.1 Overview

The following chapters present the design project. It is organised in three chapters. This chapter (four) presents the user perspective of the Virtual Private Branch Exchange (VPBX). This is done with scenarios and use cases, and gives an understanding of how the system should work, from a user point of view.

Chapter five lists the functional requirements of the system. It presents the functions that the VPBX, the telephone queue, the voice mail and the client applications should have.

Chapter six is a design proposal that is made from an architectural point of view. The design is built from the requirements and presents the different modules of the VPBX and how they interact with each other, and with the JAIN framework. An evaluation of the design is discussed in chapter seven.

4.2 User categories

I have used some categories to define the different users in the system.

Attendant

The attendant is the operator or telephonist who takes the incoming calls, asks the caller who he/she wants to talk to and switches the call to the correct destination.

User

The users are all the people at the company that is connected to the VPBX, i.e. all the people that the attendant can switch the calls to.

Caller

The caller is the person calling to the company. The caller wants to talk to a user.

In figure 4.1 on the next page, a flowchart is presented that shows the different actions and choices the system and users does when a call comes in to the Virtual PBX.

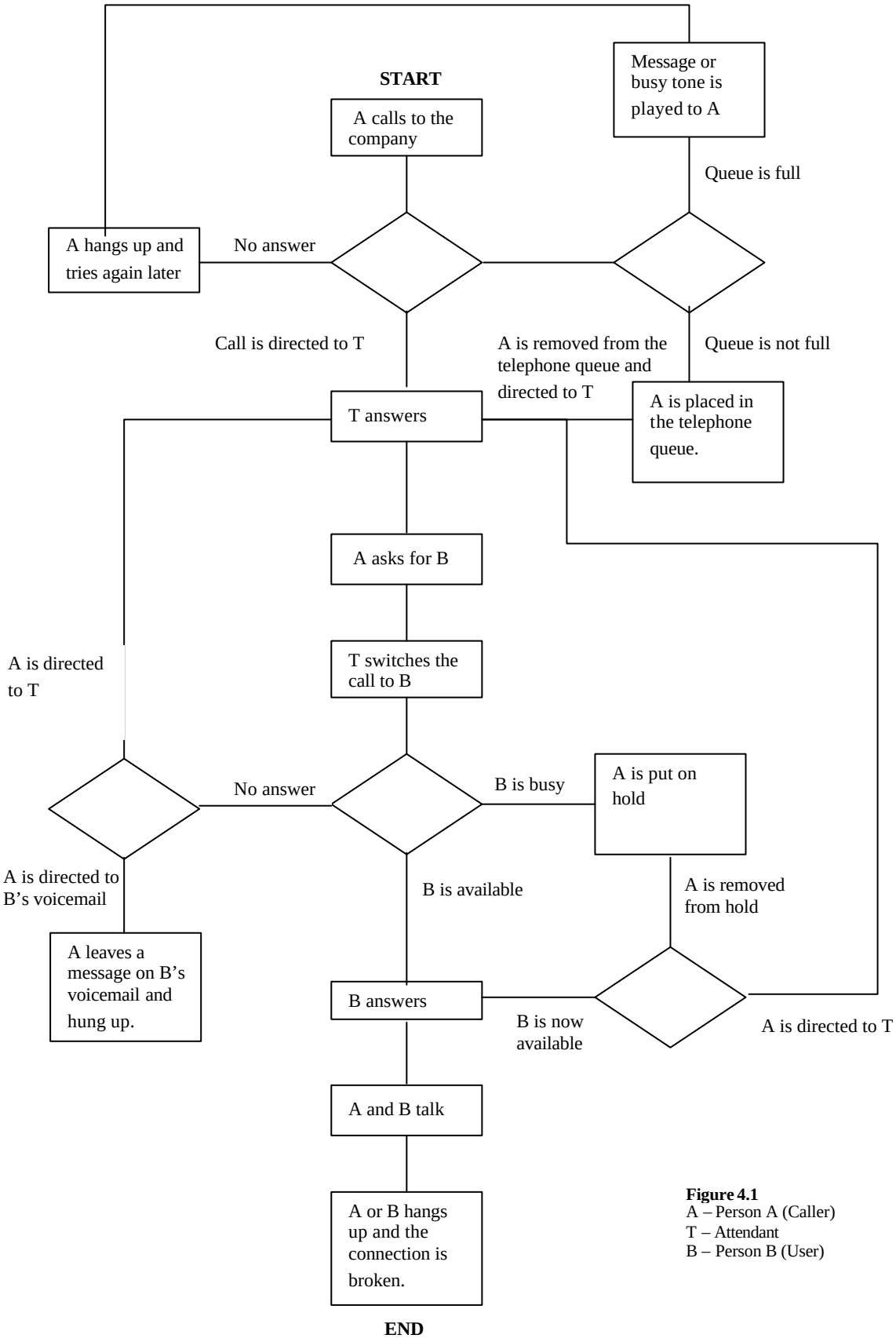

Figure 4.1
A – Person A (Caller)
T – Attendant
B – Person B (User)

4.3 Use Case 1 (Normal)

Person A wants to talk to person B and calls the company where B works. Attendant T answers and directs the call to B. This use case follows a straight line from START to END in figure 4.1.

1. A calls the company number.
2. Attendant T answers the phone.
3. A asks for B
4. T switches the call to B.
5. B answers the phone.
6. The call between A and B is established

Alternatives:

2b. If the attendant T don't answer the phone.

5b. If B is busy, the call should go back to T. T should be able to put A on hold. When B is available again, A should be removed from hold and directed to B.

4.4 Use Case 2 (Caller is put on telephone queue)

Person A wants to talk to person B and calls the company where B works. Attendant T answers and directs the call to B. At the same time person C calls to talk to person D. Since T is busy, C is put on the telephone queue. When T is available C is released from the queue. The use case can be followed in figure 4.1.

1. A calls the company number.
2. Attendant T answers the phone.
3. C calls the company number
4. Since T is busy, C is put on the telephone queue
5. A asks for B
6. T switches the call to B.
7. B answers the phone.
8. The call between A and B is established
9. C is removed from the queue and T answers
10. C asks for D
11. T switches the call to D
12. D answers the phone
13. The call between C and D is established

Alternatives:

4b. Any person calling to the company should be placed in the telephone queue if the queue is not empty or the attendant is busy.

4c. If the telephone queue is full, then C should get a busy tone.

6b. If B is busy; T should be able to put A on hold. When B is available again, A should be removed from hold and directed to B. This should be done automatically.

11b. Same as 6b, if D is busy, C should be put on hold.

4.5 Use Case 3 (Caller is put on hold)

Person A wants to talk to person B and calls the company where B works. Attendant T answers and directs the call to B. Since B is busy, A is put on hold. B continues to be busy, so after a while T takes back the call to check with A if he/she still wants to hold. The use case can be followed in figure 4.1.

1. A calls the company number.
2. Attendant T answers the phone.
3. A asks for B
4. B is busy so T puts the call on hold.
5. Some time passes while B is still busy. I.e. A is on hold.
6. The call is switched back to T.
7. T asks A if he/she wants to keep holding.
8. A is put back on hold.
9. B is now available
10. A is removed from hold and switched to B.
11. B answers the phone
12. The call between A and B is established

Alternatives:

8b. A does not want to keep holding, but hangs up. End of use case.

5 FUNCTIONAL REQUIREMENTS

5.1 Virtual PBX

Incoming calls:

- The VPBX should be able to take incoming calls to the company.
- Incoming calls should be sent to the telephone queue if the queue is not empty.
- Incoming calls should be sent to telephone queue if the attendant is busy or not available.
- If the telephone queue is full a message or busy tone should be played to the incoming call.
- If the telephone queue is empty and the attendant available, then the call should directly and automatically be routed to the attendant and not to the queue.

Route calls:

- The VPBX should be able to route calls to the attendant
- The VPBX should be able to get call from telephone queue and route call to attendant.
- VPBX should be able to take call from attendant and re-route the call to correct user, on request from attendant.

Telephone queue

- VPBX should be able to put call on the telephone queue
- VPBX should be able to get call from the telephone queue
- VPBX should be able to get information about the queue from the telephone queue (no of calls on the queue, etc)

5.2 Telephone queue

- Organize calls on a telephone queue according to the first in – first out (FIFO) principle.
- Be able to put calls on the queue
- Be able to remove calls from the queue
- Be able to delete calls from the queue, if the caller hangs up or the connection is broken.
- Be able to play messages to the caller
- Be able to play music or ring tone to the caller
- Send information about the queue to VPBX: Information should be (no of calls on queue)
- Handle requests from VPBX such as (Get call from queue)

5.3 Voice mail

- All users should have a voice mail
- Users should be able to record a voice mail message
- Users should be able to listen to messages in the voice mail by dialling a short number, for example 133.
- Callers should be directed to the users voice mail if the user is not available.

5.4 Attendant Client Application

The attendant should get information from the VPBX and send requests to the VPBX. The attendant should be able to do this via an application in the mobile phone.

Information sent from the VPBX to the attendant client application:

- Number of calls in telephone queue
- Status of the users (mobile phone ON, OFF or BUSY)
- Other user information (user is available, in meeting, gone for the day, etc)
- Information about the caller (Name, number) if available.
- Call on hold

Request options from attendant client application to VPBX:

- Route calling party to destination D.
- Put calling party on hold for destination D.
- Get calling party on hold from destination D.
- Put calling party on hold (attendants own hold).
- Get calling party from hold (attendants own hold).
- Get next call from queue

User interface:

The following should be available in the user interface for the attendant client application:

- Show number of calls in queue
- Show a list of the users (employees) with name, number, status and other information available.
- Show calling party call ID (phone number)
- Show if user has call on hold and the time this call has been on hold.
- Attendant should be able to search the list of users by name, short number or other information.

Others:

- The attendant client application should have a data connection to the VPBX

5.5 User Client Application

- A user should be able to tell the VPBX when the user is not available for calls by giving the VPBX the following information (activity, from, to), where from and to are date and time of the activity
- Activities can be: meeting, lunch, gone for the day, sick, vacation, away, etc
- A user should be able to add a new activity.
- A user should be able to edit an existing activity
- A user should be able to delete an existing activity
- The user client application should be accessible from a mobile phone.
- An Internet based web version of the user client application should also be available and accessible with an ordinary web browser like Internet Explorer.

6 DESIGN PROPOSAL

6.1 System Overview

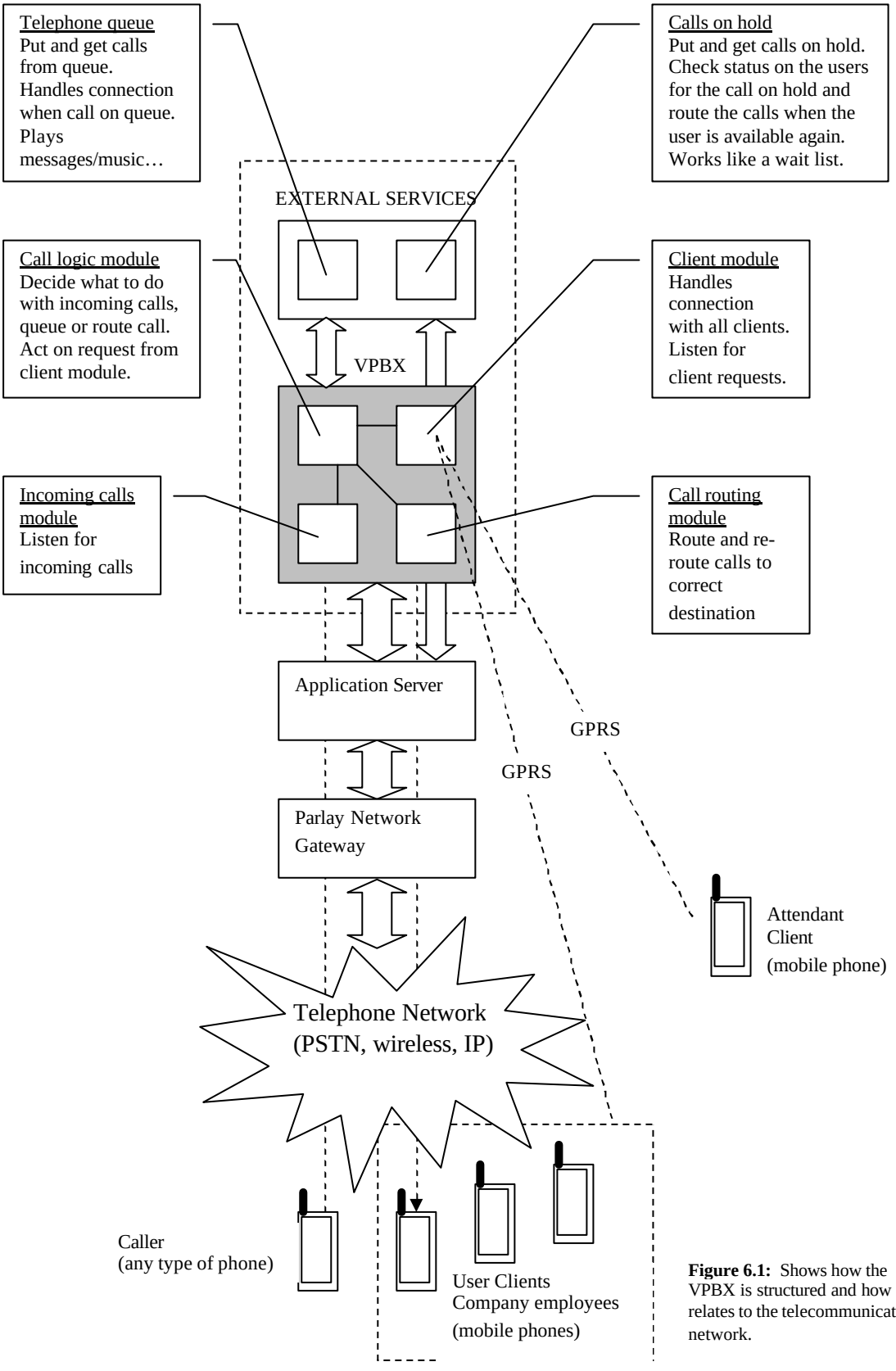

Figure 6.1: Shows how the VPBX is structured and how it relates to the telecommunication network.

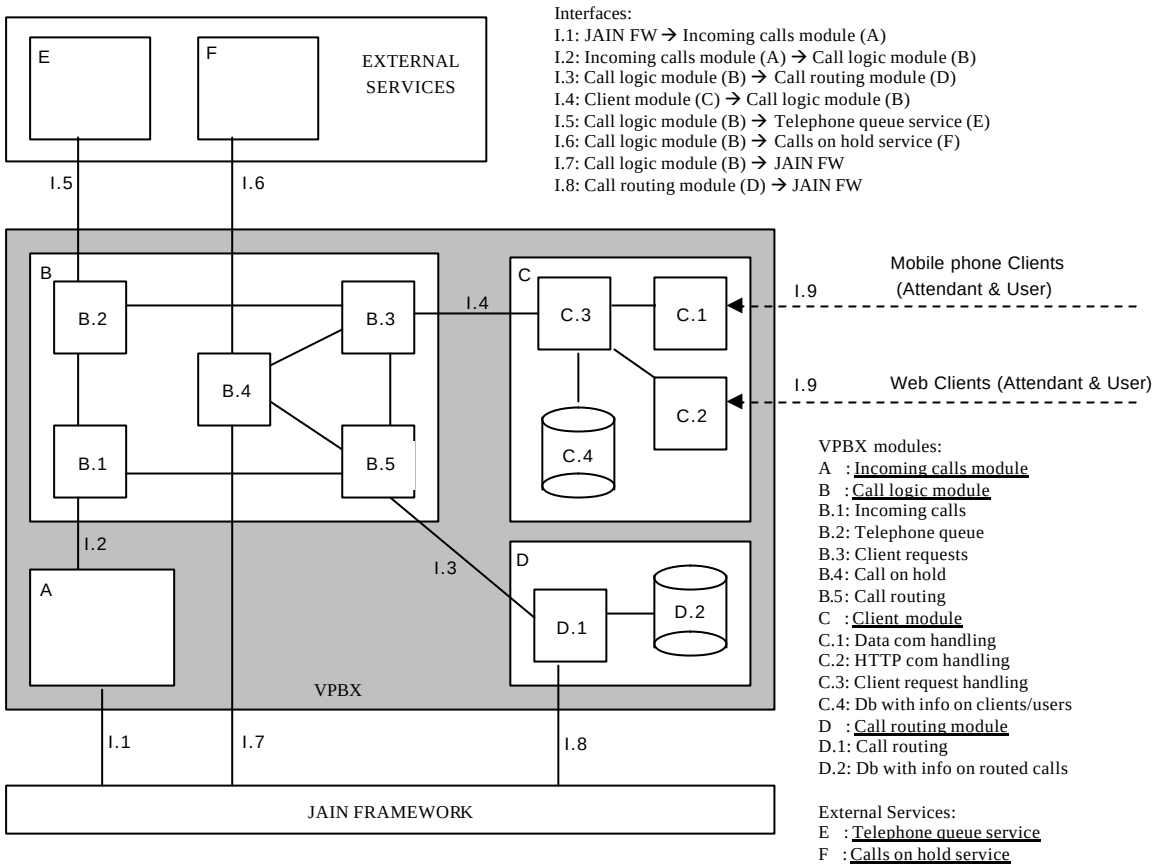

Figure 6.2: Overview of modules and interfaces in the VPBX.

6.1.1 Modules overview

As seen in figure 6.1 and figure 6.2 the VPBX is divided into four modules (A, B, C and D), which are also divided into smaller sub-modules (B.1, B.2 etc). Two external services (E and F) are also shown in the figure as well as the interfaces (I.1, I.2 etc) between the modules. This section will present the modules and the next section will present the interfaces.

Incoming calls module

This module listens for incoming calls from the JAIN framework. By listening for a callback from the framework it will detect when an incoming call is coming from the network. This module then takes over and creates a call object that is the main object that handles a call. When the call object is created it turns the call over to the Call logic module.

Call logic module

The call logic module makes decisions about what to do with a call. It does the decision making for the VPBX. This module is quite big and is therefore divided into smaller sub-modules. The most important sub-modules are:

- Incoming calls (B.1)
- Client requests (B.3)
- Calls on hold (B.4)

Incoming calls

This sub-module is invoked from the incoming call module. It takes a call object and decides whether to route the call to the attendant or to put the call on the telephone queue.

Client requests

This sub-module executes or forwards (in the case of call on hold) the request or instruction that comes from the client module. The most important requests are: get next call from queue, route call to destination, put call on hold, and get call on hold.

Call on hold

This sub-module executes when a request comes from the client module to put or get a call on hold. The sub-module communicates with the JAIN framework to check status on the mobile phones to which it has calls on hold for. When a mobile phone changes status from busy to available, the sub-module transfers the call to the call routing module (D) to route the call.

Other sub-modules

The other sub-modules in figure 6.2, B.2 and B.5 includes very little logic and is more or less just implementations of the interfaces to the telephone queue service (E) and the call routing module (D).

Client module

In figure 6.2 the client module (C) is divided in some smaller sub-modules and a database. Sub-modules C.1 and C.2 handles the actual data communication with the clients, both attendant client and user clients. The difference is what communication protocol they support. C.1 supports an interface to J2ME clients on mobile phones. C.2 provides a HTTP interface for web clients. C.3 is the sub-module that is the decision maker in the client module. It checks authentication on clients, updates the database with status on the users and forwards requests from the attendant client to the call logic module (B). The database C.4 has information about the users (i.e. the company employees), their status and requests.

Call routing module

This module (D) performs the actual routing of a call. It communicates with the JAIN framework through the JAIN SPA to route the calls. This module also keeps track of ongoing calls and handles broken connections, terminates call objects and updates the call database. The module has a database where it stores the call objects and information related to them, so that actions can be taken when a request to re-route a call is made. The module communicates very much with the JAIN framework and register call back methods on the calls, so it will be notified on changes on ongoing calls.

6.1.2 External services

Two external services are used by the VPBX: Telephone queue and calls on hold. The internal architecture of these services is out of the scope of this report, but I will give a short description of them here.

Telephone queue service

The telephone queue service (E) handles calls that are not routed to a destination yet. It stores the calls in a data structure in the First in – First out order. When a call is in the queue the service can play music or messages to the caller, such as for example: “You are now number 4 in the queue.”

The Call logic module (B) in the VPBX transfers a call to the telephone queue, when the call shall be put on the queue. When a call is in the queue, the telephone queue service has the ownership of the call. When a request to get next call on the queue comes from the Call logic module, the call is transferred back to the VPBX and is no longer owned by the telephone queue service.

Calls on hold service

The call on hold service (F) is a wait list for calls that is to be routed to a specific destination, which is busy at the time. The call is therefore put on hold. It has a data structure in which is stores the calls, but all logic such as tracking status and deciding when to route the call is made by the Call logic module (B) in the VPBX.

6.1.3 Interface overview

Following is a listing of the interfaces that are shown in figure 6.2 (as I.1, I.2 etc), with their most important methods.

JAIN FW / Incoming Calls module (I.1)

Methods from the Incoming calls module to the JAIN framework:	
`authenticateClient()`	Called to authenticate the VPBX with the JAIN framework.
`signServiceAgreement(serviceID, props)`	Called from the VPBX to the JAIN framework to establish a service agreement on which services in the JAIN framework, the VPBX should use and have access to.
`requestCall(aCallLeg)`	Called when VPBX has been notified on an incoming call. This method hands over the call from the JAIN framework to the VPBX.
Methods from the JAIN framework to the Incoming calls module:	
`authenticateFramework()`	Called to authenticate the JAIN framework with the VPBX.
`notifyIncomingCall()`	Notifies the VPBX on an incoming call.

Incoming Calls module / Call Logic module (I.2)

Methods from the Incoming calls module to the Call logic module:	
`IncomingCall(callObj)`	Incoming Calls module calls this method in the Call logic module after it has got an incoming call, created call object and call legs for the caller (origin). The method turns over a call object with caller set.

Call Logic module / Call Routing module (I.3)

Methods from the Call logic module to the Call routing module:	
`routeCall(callObj)`	Called when a new Call should be routed. A call object with call legs for both calling party and destination party should be passed as a parameter.
`getCall(id)`	Is called when the call logic module want to get back a call that is already routed to a destination. The call routing module then releases the destination and returns the call to the call logic module.

Client module / Call Logic module (I.4)

Methods from the Client module to the Call logic module:	
`getNewCallFromQueue()`	Called when the attendant client has made a request to get a new call from the telephone queue. Call logic module gets the call from the telephone queue service, attaches a call leg to the attendant and passes the call over to the call routing module to route the call.
`putCallOnHold(dest)`	Called when the attendant client has made a request to put the ongoing call on hold. Parameter dest is the destination (i.e. the user or mobile phone number), where the call should be routed. The call logic module routes the call automatically when the user gets available again.
`getCallOnHold(userID)`	Called when the attendant client has made a request to get back a call that is put on hold.
`routeToUser(userID)`	Called when the attendant client has made a request to route the ongoing call to another user.
`getNumberOfCallsInQueue()`	Returns the number of calls that are in the telephone queue.
Methods from the Call logic module to the Client module:	
`updateUserStatus(status)`	Called when the call logic module want to update the attendant client with changes in user status.

`updateCallerInfo(info)`	Called when the call logic module want to update the attendant client with changes on the caller.

Call Logic module / Telephone queue service (I.5)

Methods from the Call logic module to the Telephone queue service:	
`getNewCall()`	Returns the first call in the telephone queue list.
`putNewCall()`	Adds a new call last in the telephone queue list.
`getNumberOfCallsInQueue()`	Returns the number of calls in the telephone queue list.

Call Logic module / Calls on hold service (I.6)

Methods from the Call logic module to the Calls on hold service:	
`putCallOnHold(callObj)`	Adds a new call in the calls on hold data structure.
`getCallOnHold(userId)`	Gets a call from the calls on hold data structure.

Call Logic module / JAIN FW (I.7)

Methods from the Call logic module to the JAIN framework:	
`getUserStatus(msisdn[])`	A call to the JAIN framework to get the status (ON, OFF or BUSY) on the users. The array msisdn[ ] includes a list of the users mobile phone numbers to be checked.

Call Routing module / JAIN FW (I.8)

Methods from the Call routing module to the JAIN framework:	
`destCallLeg.release()`	A call to the JAIN framework to release a specific call leg. The destination is disconnected from the call.
`CallLegOrig.route()`	A call to the JAIN framework to route the originating call leg.
`CallLegDest.route()`	A call to the JAIN framework to route the destination call leg.
`CallLegDest.attachMedia()`	This method requests that the call leg be attached to its call object. This will allow transmission on all associated bearer connections to and from other parties in the call.

Clients / Client module (I.9)

Methods from the Attendant client to the Client module:	
`login(userID,pw)`	To login to the system
`logout(userID)`	To logout of the system
`getNewCallFromQueue()`	A request from the attendant client to get the next call in the telephone queue.
`putCallOnHold(dest)`	A request from the attendant client to put the ongoing call on hold for a specific destination.
`getCallOnHold(userID)`	A request from the attendant client to get back a call that is put on hold for a specific user.
`routeToUser(userID)`	A request from the attendant client to route the ongoing call to a specific user.
`getUserStatus(users[])`	A request from the attendant client to get the status on some users.
`getCallerInfo()`	A request from the attendant client to get information about the originating caller of the ongoing call, that is set up to the attendant.
`getNumberOfCallsInQueue()`	A request from the attendant client to get the number of calls in the telephone queue.
Methods from the Client module to the Attendant client:	
`updateUserStatus(status)`	Information about users' status is pushed to the attendant client and updates the local user status information in the client.
`updateCallerInfo(info)`	Information about the caller is pushed to the attendant client and updates the information in the client.
`updateNumberOfCallsInQueue(no)`	Information about the number of calls that are in telephone queue is pushed to the attendant client and updates the information in the client.
`updateAll(all)`	A single method that performs the same as: `updateUserStatus()`, `updateCallerInfo()`, and `updateNumberOfCallsInQueue()`
Methods from the User client to the Client module:	
`login(userID,pw)`	To login to the system

`logout(userID)`	To logout of the system
`createActivity(userID, type, from, to)`	Creates an activity in the user's calendar. type is type of activity (for example "meeting"), from and to are date and time.
`deleteActivity(id)`	Deletes an activity item.
`updateActivity(id)`	Updates an activity item.
`getCalendar()`	Returns the users calendar with all activities.
`getCalendar(date)`	Returns the users calendar or schedule for a specific day.

6.2 Clients

As described before, two different types of clients exist: users and attendants. The attendant is the operator who takes the calls and directs the calls to the correct destination. The users are the other people in the company to whom the calls are directed.

6.2.1 User client

The user client presents a user interface to the users and a connection to the VPBX. The client makes it possible for users to tell the VPBX if the user is available or not reachable. The user can in the client application add activities in a schedule when the user is not reachable, and here give a reason. The method call is shown in figure 6.3. Activities are as described in the requirements section, meeting, lunch, gone for the day, etc.

The user should have at least two different ways to edit these activities. One should be via the mobile phone. The technique could be either by a Java client based on J2ME or a WAP-based application. Both techniques give a good interface for the user. The bandwidth of the data connection between the mobile phone and the VPBX is preferably similar to GPRS or better. At the time of writing, GPRS is the best choice available for this data connection.

The second way to access the VPBX should be via a web page where the user easily can edit activities and get a good overview of the schedule, both on days and weeks.

A third way is possible but not necessary, where the user can dial a telephone number to the VPBX and via a voice or number code system edit the activities.

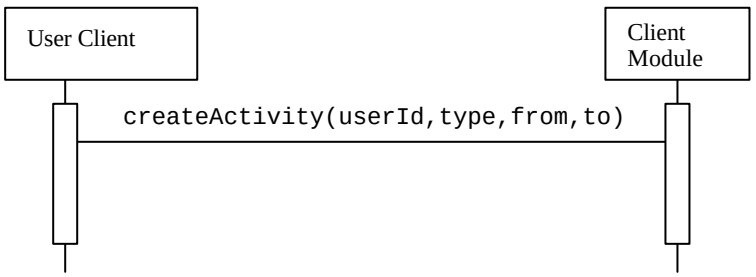

Figure 6.3: Shows how the user can tell the client module when the user is not reachable.

6.2.2 Attendant client

The attendant client needs to be a bigger application than the user client. The attendant will need to have a lot more information presented in the user interface. This includes the number of calls in telephone queue, the number of the calling party, a list of users or a way to search among the users. Beside this it need to have functions for the attendant to get new calls from the telephone queue, switch a call to a specific user, and put calls on hold. Further more it needs to display status information about the users; such as if the user is busy, at a meeting, etc. To be able to present all this information to the attendant in a good way, the attendant client needs to be a lot bigger and more complex than the user client.

I see two main techniques for this client. The first one is a Java based J2ME application which can be run in a mobile phone. Since the application needs to present a lot of information in the user interface, a mobile phone with a large screen is necessary. Also preferable is a touch screen, to enable fast and efficient access to the functions. The same requirements to the bandwidth of the data connection as discussed on the user client should be met. For the attendant client it is even more critical that the connection is fast and efficient. Therefore a GPRS or better is a requirement.

The second technique for the attendant client should be a web based client, in which the attendant more easily can access all the functions. A list of the users with information about their status and easier ways to search the user list will be presented.

Since the purpose of this product is to enable full mobility of the personnel, even the attendants, a client made for a mobile phone should be present. People are not always up and running, so therefore a good and complete client should also be given via a website. The choice of a website versus another type of application is to maintain mobility and flexibility as much as possible. With a web-based solution, it is possible to access the client on any computer; the only thing that is needed is an Internet connection and a web browser.

Even if the attendant is working the web-based client, all calls are made to the attendant's mobile phone. The call could still be controlled from the website since the system is server based and the client only is an interface to the system.

6.3 Virtual PBX

6.3.1 Registration in the JAIN framework

Before the VPBX can work properly towards the telecom network, it needs to register with the JAIN framework or more specifically with the Parlay Gateway. As seen in figure 6.4 this is done in an authorization process were the VPBX authenticates itself towards the Parlay Gateway, and the Parlay Gateway authenticates itself toward the VPBX. The authentication process is done through the JAIN Service Provider API (JAIN SPA).

When the VPBX and the framework have authenticated each other, they have to set up a service agreement for the services in the framework that the VPBX will use. The services that will be used by the VPBX are: multiparty call control, user interaction and user status. Multiparty call control for call related functionality such as creating calls, routing calls and deleting calls. User interaction to enable functions for playing messages and music to the caller. User status; to check for status (on, off, busy) on mobile phones.

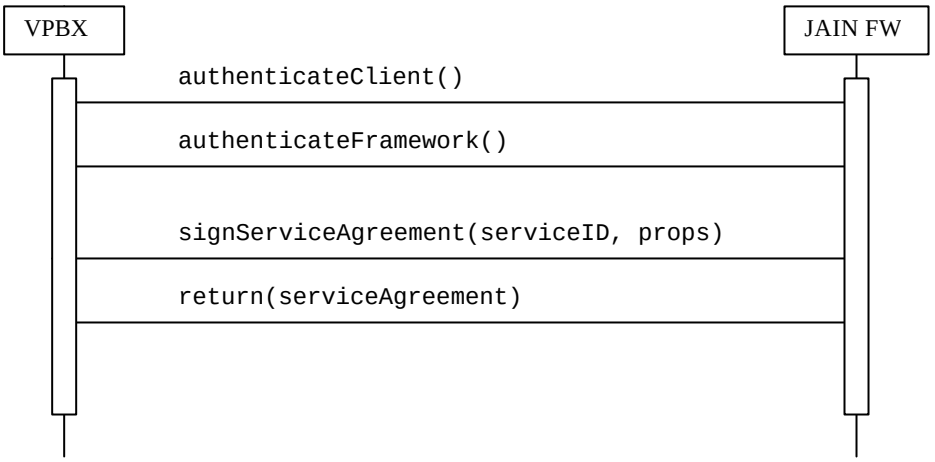

Figure 6.4: The VPBX and the JAIN framework authenticates itself and signs service agreements for the services to use.

6.3.2 Incoming calls module

Figure 6.5 shows how this module listens for incoming calls from the JAIN framework. By listening for a callback from the framework it will detect when an incoming call is coming from the network. This module then takes over and creates a call object that is the main object that handles a call. After that it creates a call leg for the calling party and connects that to the call object. To be able to route a call, the call has to have two or more call legs. This module does not create the destination call leg since it only handles incoming calls. Instead it passes the call object over to the call logic module to decide what to do with the call.

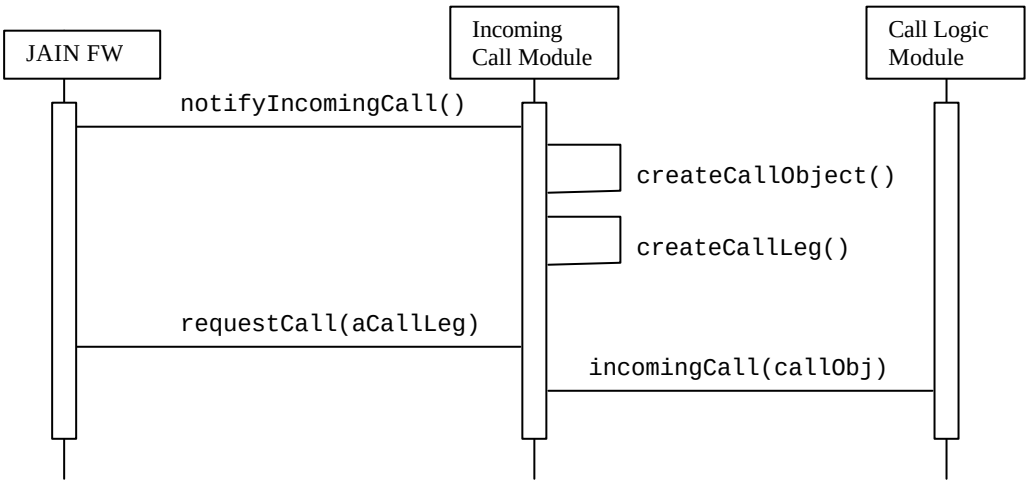

Figure 6.5: A new incoming call is detected by the JAIN framework and handled by the incoming call module.

6.3.3 Call logic module

The call logic module makes decisions about what to do with a call. It does the decision making for the VPBX. This module is a lot bigger than the other modules and is therefore divided into smaller sub-modules. The most important sub-modules are:

- Incoming calls
- Client requests
- Call on hold

Incoming calls

This sub-module is invoked from the incoming call module. It takes a call object and decides whether to route the call to the attendant or put the call on the telephone queue. If the call is to be routed to the attendant, it creates a destination call leg to the attendant and passes the call to the call routing module. If the call should be put on the telephone queue, then it passes the call to the telephone queue instead.

Call on hold module

This sub-module executes when a request comes from the client module to put or get a call on hold. The sub-module also communicates with the JAIN framework to check status on the mobile phones to which it has calls on hold for. When a mobile phone changes status from busy to available, the sub-module transfers the call to the call routing module to route the call. This is done automatically without a request from the attendant client. This sub-module also notifies the client module on changes in status.

Client requests

This sub-module executes the request or instruction that comes from the client module. The following scenarios and diagrams present the sequence of method calls for the different requests that are listed in the requirements section 3.3 above.

Handle incoming call.

A new call is detected and handled by the incoming call module. As seen in figure 6.6 the call is then sent to the call logic module, which decides what to do with the call. The incoming calls sub-module calls the telephone queue sub-module to check if the telephone queue is empty. If the telephone queue is not empty, other calls are already on the queue, and the call is therefore also put on the queue.

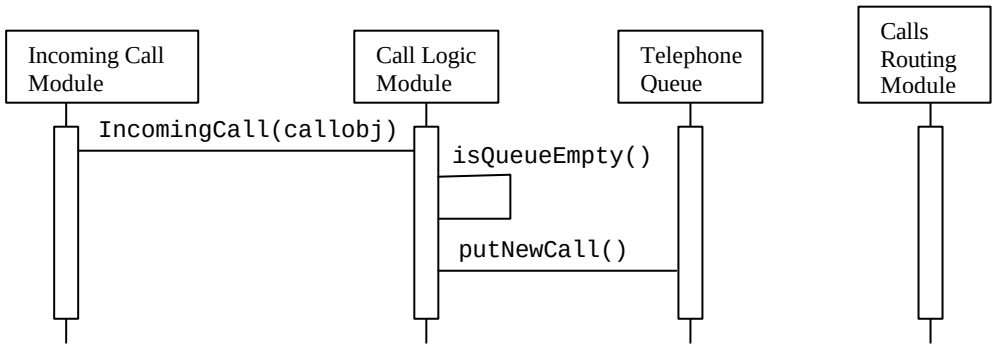

Figure 6.6: A new incoming call is sent to the call logic module. Since the telephone queue is not empty the call is put on the queue.

If the telephone queue is empty, another decision is made by the call logic module. Figure 6.7 shows the scenario when the queue is empty and an attendant is available. The call logic module checks if the attendant is available. If the attendant is not available, the call should be put on the telephone queue like the example above in figure 6.6. If instead the attendant is available the call should directly be routed to the attendant as shown in figure 6.7.

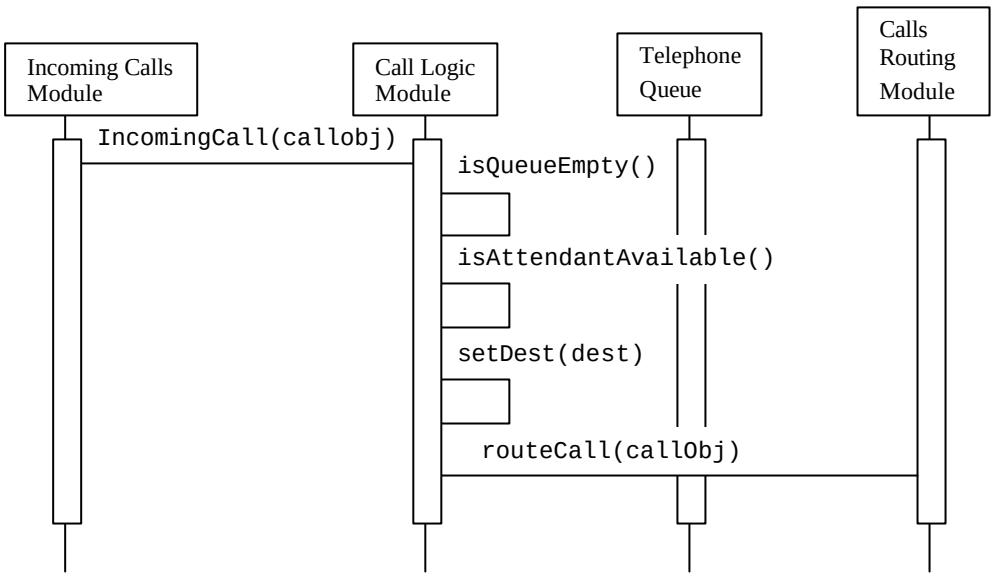

Figure 6.7: If the telephone is empty and the attendant is available, the call should directly be routed to the attendant.

Get call from telephone queue.

A request to get a new call from the telephone queue is made from the attendant client. The Call Logic Module gets the call object from the telephone queue and creates a call leg to the attendant, in the method setDest(dest). The call object is then passed to the Call Routing Module, which communicates to the JAIN framework and routes the call. The Call Logic Module then sends information about the caller such as telephone number to the Client Module, which pushes the information out to the attendant client application.

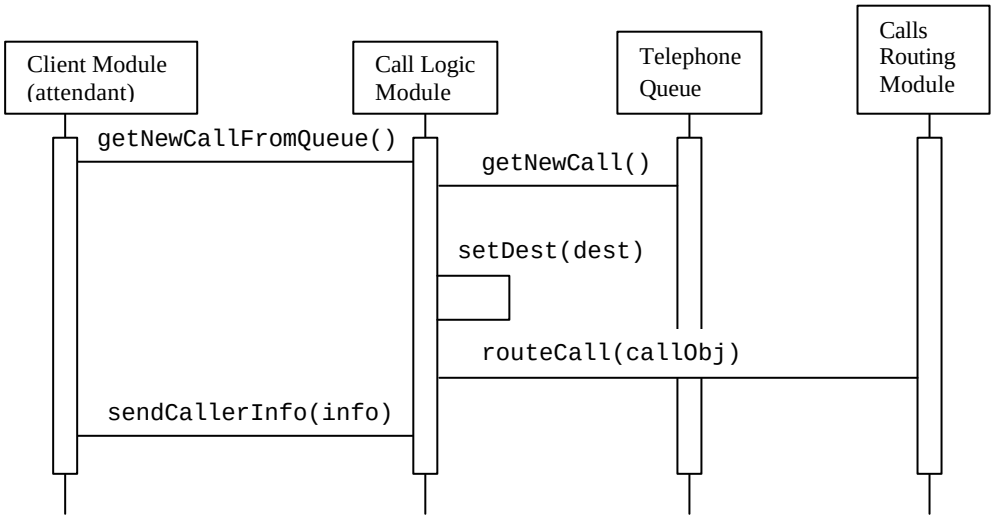

Figure 6.8: A request to get the next call on the telephone queue is made from the attendant client. The call is returned from the telephone queue, the new destination to the attendant is set in the call object, and then the call is routed to the attendant. The attendant client is also updated with information about the caller.

Put call on hold for user

When the attendant client requests to put a call on hold, the following scenario happens. The Call Logic Module makes a request to the Call Routing Module to get back the call, which is already set up between the caller and the attendant. The Call routing module then releases the call leg to the attendant and returns the call object to the call logic module. The call object is then passed to the sub-module that handles calls on hold. This sub-module takes over the call and checks for the status of the user.

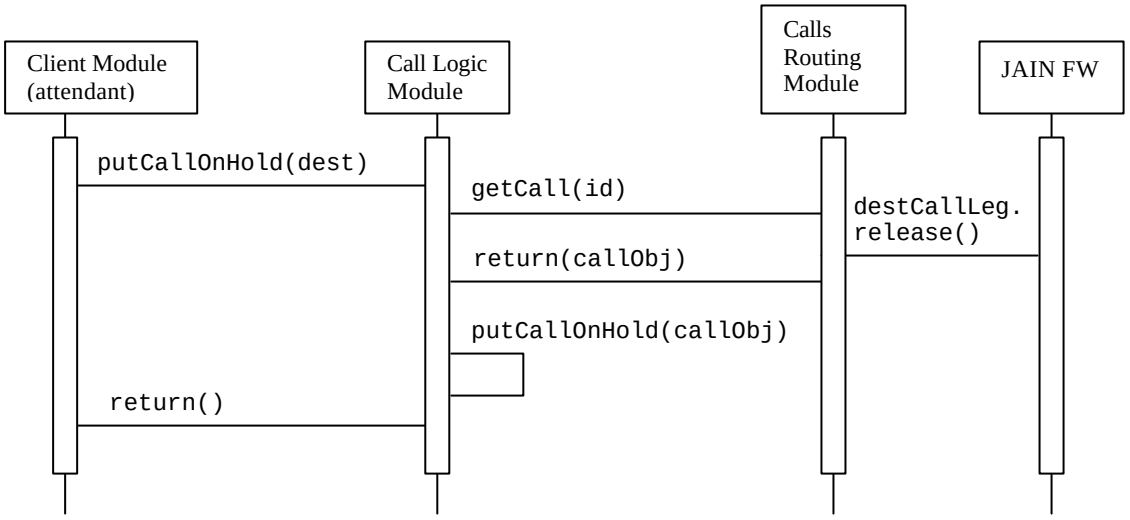

Figure 6.9: A request to put the caller on hold is made from the attendant. The call is then released from the attendant and returned to the call logic module, which put the call on hold for a specific user.

When the user is available again, the call on hold sub-module creates a new call leg to the destination user and attaches this to the call object. This is done in the method setDest(dest). The call object is then passed to the call routing module to be routed to the user. The call is now automatically routed to the user and then the attendant client is updated with a method call to the client module. This process is made automatically and not on command by the attendant client.

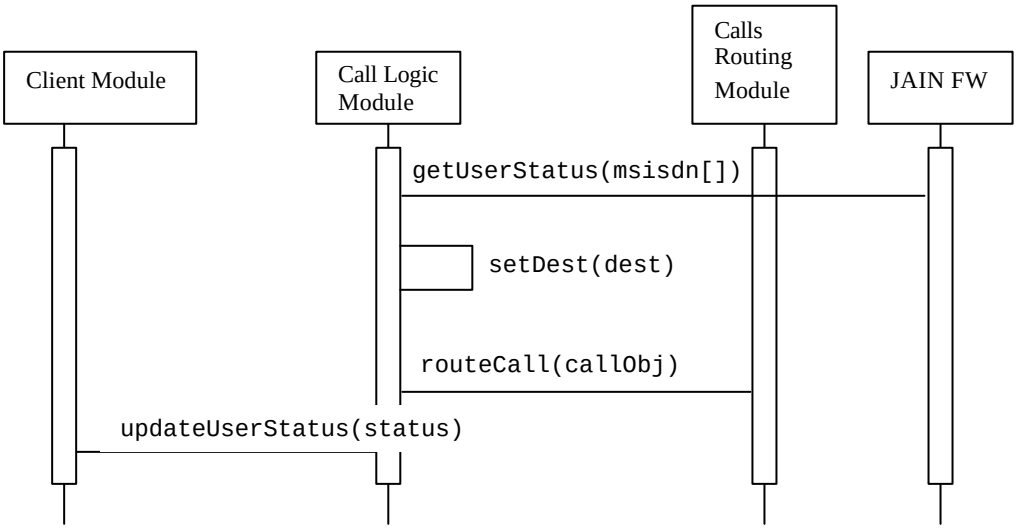

Figure 6.10: The call logic module checks the status on the users and is therefore noticed when the user is available again. The call is then routed to the user without any communication with the attendant. The attendant client is however updated with the new user status.

Get call from hold for user

A method call to get a call from hold is sent by the client module. The request is made for a specific user, which is passed as a parameter. The call object is returned from hold by a request from the call logic module. A new call leg to the attendant is created in the setDest method and then the call is routed to the attendant by a call to the call routing module, which handles the connection with the JAIN framework.

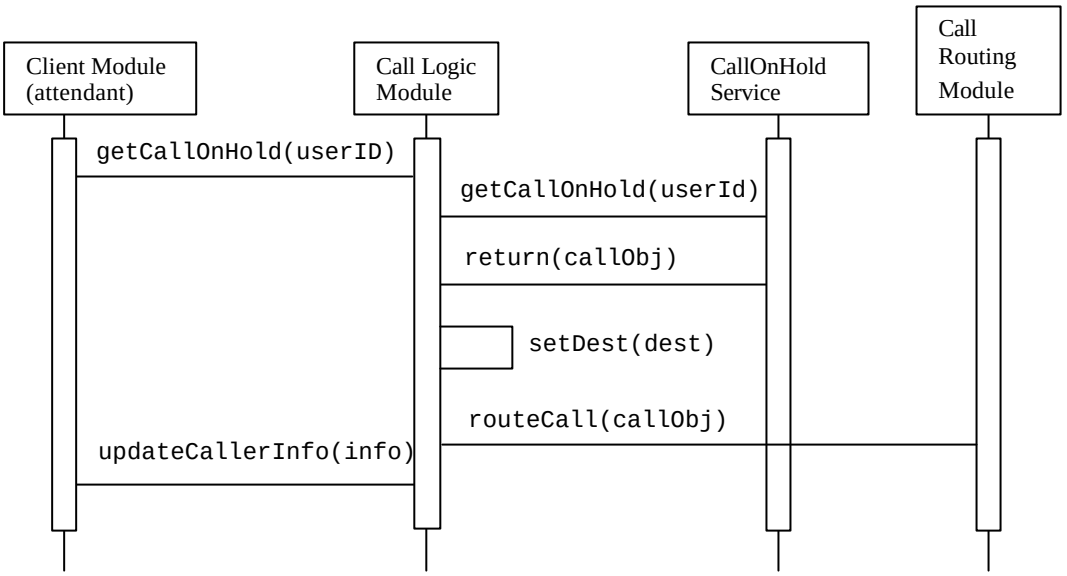

Figure 6.11: A request to get a call from hold is made by the attendant. The call logic module gets the call back and re-routes it to the attendant.

Route calling party to user

A request to route the call from the attendant to a certain user is made from the attendant client to the call logic module. The process is the same as for other scenarios. The call leg to the attendant is released and the call object is returned to the call logic module. Then the destination is set and a new call leg is created to the user. After that the call object is passed to the call routing module to be routed. The call routing module is connected to the JAIN framework and handles the actual routing of the call. This is described later in this chapter.

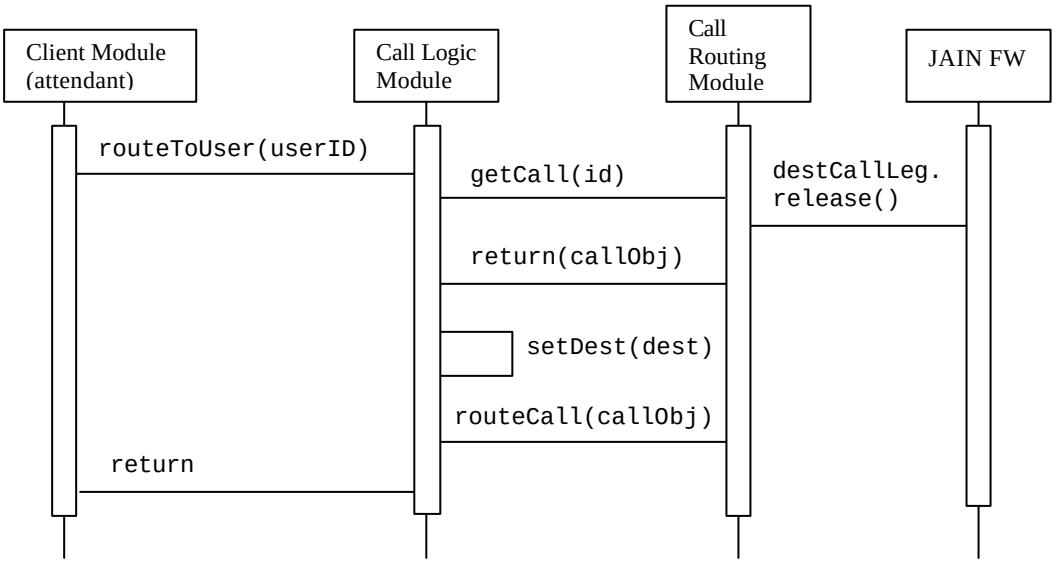

Figure 6.12: The attendant makes a request that the call should be routed to a user. The call is released from the attendant, the new destination is set on the call object, which is then routed to the user.

6.3.4 Client module

This module handles a data connection with the clients. It handles all connection related tasks and listens for requests from the clients. It also pushes information out to the clients. Two different types of clients exist, attendant client and user client.

Information sent to the attendant client:

- Number of calls in queue
- Phone number of calling party
- Status of all users (on, off, busy)
- Additional information about users (available, meeting, etc.)

The information above is pushed to the client and should therefore always be updated in the client.

Requests made from attendant client:

- Login
- Logout
- Get new call from queue
- Route call to user D
- Put call on hold for user D
- Get call on hold for user D
- Put call on hold
- Get call from hold

Login and logout is handled totally by the client module. All other requests are passed to the client requests sub-module in the call logic module, to be executed by that module. Scenarios for all requests but the login and logout are presented in earlier sections of this chapter.

Requests made from user client:

- Login
- Logout
- Create new activity
- Update activity
- Delete activity
- Get Calendar

The user client makes it possible for users to tell the VPBX if the user is available or not reachable. The user can in the client application add activities in a schedule when the user is not reachable, and here give a reason. The activities such as meeting, lunch, gone for the day, etc are described earlier in the requirements section,

The client module has a database in which it stores information about the users. It stores information such as name, password, phone number, user status, activities etc.

The client module also handles all communication with the clients. It authenticates the clients on login and listens for requests from the clients.

6.3.5 Call routing module

This module performs the actual routing of the call. When a call object is passed to this module it should have two call legs, one to the calling party and one to the destination. This module then takes care of the call and calls the appropriate methods in the JAIN SPA to route the call as seen in figure 6.13.

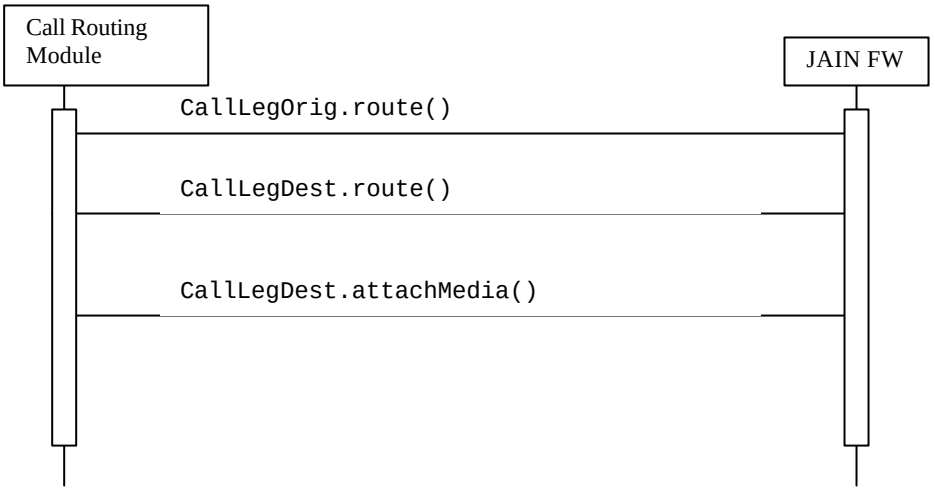

Figure 6.13: To route a call several method calls to the JAIN framework needs to be made. The most important are these three methods. The two call legs needs to be routed separately and then a call to connect the legs are made.

The call routing module has a database in which it stores the ongoing calls and information regarding these calls. This is done in order to be able to safely delete call objects when broken connections or other errors occur. This module has the ownership of the call after the call is established between the calling- and the destination-party. It is responsible to tear down the call when either side hangs up.

6.4 Voicemail

Voicemail is a service that is more or less a standard in all PBX systems today. It is therefore necessary for the VPBX to support it. A decision to make here is whether to create a new voicemail system for the VPBX system or to use the existing voicemail system that the network operator has.

Since one rule when designing this system has been to keep it as simple as possible, I have made the decision to use the voicemail system presented by the network operator. The drawback to this is that the VPBX system cannot control the functions available in the voicemail or directly link the VPBX to the voicemail. The gains are on the other hand that a voicemail already exist and is therefore not necessary to develop. The voicemail system is working well and has the necessary functions that the user needs.

7 EVALUATION OF DESIGN PROJECT

The focus when designing the VPBX system has been to create a small and flexible system. I started working on the project with the intention of designing and implementing a small software based PBX system. I misjudged the complexity and largeness of such system and have therefore not been able to implement any part of the system. Testing such a system also needs a real or simulated telecom network, which I didn't have access to. I could therefore not implement and test even smaller parts of the system. My intention of the design was to create a detailed and thorough design that covered all aspects. This was neither possible since the time scope for this project was only ten weeks. I have had to settle with an architectural design on a higher level. This architectural design however gives a good starting point for further and deeper work on this virtual PBX system.

7.1 Functional Requirements

The functionality that I focused on (listed in functional requirements, chapter 5) was a PBX system for mobile phones, with full telephone queue handling and support for the attendant to manage the system through a client application in his/her mobile phone.

I believe I have met the most important functional requirements in the design. All requirements about the VPBX such as incoming calls, route calls, put calls in the telephone queue and put calls on hold are met. I have also covered the client requests and interfaces between the VPBX and the clients, both the attendant and user clients.

I have not created any internal design on the telephone queue or the calls on hold services, this because of three things.

First these services are complex in them selfs since they need to establish a connection with the caller and play dial tones, messages or music to the caller when the caller is put on hold or placed in the telephone queue.

Second, it is possible you can use an existing telephone queue system from a third party developer and therefore only the interface towards the system is interesting.

And third I didn't have time to focus on the whole system and chose therefore to leave them out of this project.

Another question was to include voicemail in the VPBX system. Since the focus was to create a small and flexible system, I decided to rely on the already existing voicemail system in the telecom network instead of creating a new integrated voicemail system.

7.2 Quality Requirements

When creating a system for voice communication the quality is extremely important. People are used to telephones and are not tolerant for any longer response times. They require instant feedback through dial tones, busy tones etc. Delays for only a couple of seconds are very frustrating and not tolerated. Maintenance will probably also be needed since a PBX has a rather long life time. These and more issues are important and therefore the system must be built with a strong focus on quality.

The evaluation of the quality requirements for this project has been hard to make since the lack of opportunitice to test the system. Therefor this chapter discusses some quality requirements that is important when building this type of product.

7.2.1 Maintainability

A system like the VPBX has a long lifetime and therefore new technology and requirements will occur during its lifetime. This creates the need to make changes to the system and/or to develop new modules or sub-modules. The system must therefore be built from the beginning with maintainability and flexibility in mind.

In the design I have tried to keep functionality isolated in the modules or sub-modules as much as possible. It should be possible to replace a module or sub-module with a new one, without to much or any changes in the other modules. I think it is important with low coupling between the modules just because of the possible changes in the future. Replacing modules or adding new modules are very believable scenarios. The system might for example need to support new technology or protocols for data communication in the future. For example GPRS will be replaced with new and faster technology; also IPv6 will introduce new requirements etc. It is therefore important that the client module can be changed to support these new features.

7.2.2 Flexibility

One important flexibility issue is if replacing a module is possible during runtime, or if the system must be taken out of operation to make the change. The VPBX system needs to be available as much as possible and therefore it is very useful if replacing modules can be done in runtime.

I do believe it is possible to create the VPBX system so that changes can be made in runtime. This should not be so difficult if flexibility is a criterium when designing and implementing the system, but it will however require a lot more work during design and implementation. In the long run it would probably pay off since it would increase the flexibility of the system by a large amount.

Another feature that would make the VPBX more flexible would be to enable more than one attendant operator. If for example three users worked as attendants, the incoming calls could be directed to the attendant that is available at the time or divided among the attendants by a round robin order. This is a feature that is easy to include and provides a great level of flexibility from the user's perspective.

7.2.3 Availability

As described before it is very important that the system always is available. People require fast response time and total availability in the system. The system must be up and running 24 hours a day, 7 days a week. People require voice communication through telephones to always work. Therefore it is important to build the system very fault tolerant, so that the system can handle when something goes wrong and return to a safe state. Several things can happen that the VPBX must take care of; such as if a call is dropped, the caller hangs up, a mobile phone looses its connection with the network, and many more scenarios.

Of course it is impossible to say if the VPBX design will enable good availability. A more detailed design and prototype need to be created first in order to find a good implementation that can provide the availability needed in the VPBX system.

Designing a fault tolerant system is very hard, but it is however cruicial when turning the VPBX into a commercial product and in fact probably the most important quality requirement.

7.2.4 Performance and scalability

The focus for the VPBX system has been to create a system for a small sized company and not an enterprise solution. The performance needs neither the less to be high, because of the requirements on voice communication which is discussed earlier.

The system is designed for a smaller amount of simultaneous calls and is therefore suitable for a smaller company. Performance also depends on the server hardware and network speed used to run the VPBX.

Running one company on one server should be possible without any performance problem, but if several companies are run on the same hardware server the work load might be too big and performance will suffer.

The VPBX system can therefore be created for two purposes. Either it runs with only one company or it runs as a PBX for several companies. The second is probably if it is run by a network operator which provides the service to many companies. These two approaches differ a lot in the amount of traffic in the system. The second approach will probably require a much higher level of quality, not only in performance, but also in availability, flexibility and maintainability.

When the workload is increasing bottlenecks usually occur whinin the system. A solution to get around this problem and provide good performance for this larger system could be to run the different modules on separate hardware servers. This increases network traffic and creates a new possible bottleneck on the communication between the different hardware servers.

Also having more than one module running in parallel might increase the performance and scalability. As the example above with different hardware servers, this example also makes the system more complex and harder to maintain. These examples are however only thoughts and not based on any mesurements or calculations.

To enable this scalability the design needs much more work and prototypes needs to be created to test the performance, when many calls are handled simultaneously.

7.2.5 Usability

If the VPBX will be useful, it needs to provide the functionality that the users need. One such feature is the flexibility from the users perspective. Different users have different needs and therefore the system must be able to handle several types of client, such as mobile phones, stationary desktop computers, laptops, etc.

Another feature is the mobility of the system. It should be possible for the attendant to use it with a mobile phone (i.e. a smartphone with a large display). This however puts great requirement on the user interface, since a phone has a rather small display, even if the display is big for a phone. A pen-based touch screen might be a good solution for such a device. A good comparison could be that it should be almost as easy to use the system for an attendant as if he/she was using a stationary computer.

8 EVALUATION OF JAIN TECHNOLOGY

In this section I will present some of the observations and experiences about Parlay and JAIN I have made during my thesis project.

8.1 What is good with JAIN?

8.1.1 JAIN Service Provider API

The JAIN Service Provider API is a well defined Java API that is rather easy to understand. To use the API you need to have some telecom knowledge, but it is much easier to develop applications for the JAIN SPA than older Intelligent Network solutions for the telecom network. Before, you needed to work with proprietary protocols and IN services that the network provider used in their networks. With JAIN this is not needed since you have the JAIN SPA as a standardized interface towards the secure telecom network.

JAIN SPA is built on well known technologies such as CORBA and the Java programming language. Since I have a good knowledge of Java I had no problem with understanding the API. My problem was my lack of deeper knowledge in telecommunications. Therefore, I have had difficulties with understanding how JAIN and Parlay works and how my applications interact with the services in the telecom network.

8.1.2 Network Provider independence

Parlay and JAIN is from the beginning created with a focus on providing independence towards network providers. No proprietary interfaces are needed since Parlay encapsulates the underlying telecom network. A Parlay gateway is used to access the telecom network in a secure manner, and the interface towards the parlay gateway is standardised by the Parlay Group. It is therefore possible to use the same application towards several parlay gateways in separate networks.

Parlay and JAIN is both very new. Market adoption of JAIN and Parlay has just started and it is therefore very difficult to say how much independent it will be. If you look at the standard and white papers about JAIN and Parlay, it looks very good, but if the network providers will provide access through Parlay gateways in the extent the Parlay supporters say is still unknown.

8.1.3 Network convergence

Another focus of Parlay and JAIN has been network convergence. Calls in the PSTN, PLMN and packet networks are treated the same way. The Parlay gateway encapsulates the underlying signaling protocols and therefore the application programmer does not need to know which network bearer the call uses. Different parties of the call can use different bearer technology and signaling protocols. To the JAIN application this does not matter.

I think this is one of Parlay's and JAIN's big advantages over other alternatives. A common opinion in telecommunications today is that the IP/ATM, PSTN and the wireless networks will merge into one network. Technologies like Parlay and JAIN is the first step towards such a solution.

8.2 What could be better with JAIN?

8.2.1 Documentation

Apart from the specifications, there is very little documentation about JAIN and Parlay. All I have found is one book about JAIN that covers the JAIN specification and describes how JAIN and Parlay fit together (Jepsen et al. 2001). Apart from that I have only found some white papers and some reports in IEEE Communications Magazine (see references). I have tried to find examples and developer tools on the Internet, but it seems to be very little written about JAIN and Parlay. This might be because Parlay and JAIN are new technologies, but they have been around since 1998 so there should be much more information I think. You can find enough information to understand the Parlay and JAIN standards, but if you want to develop applications for JAIN you can not find example code and such things on the Internet.

The documentation on JAIN and Parlay are also mostly positive to the technology. It is only a few authors that has written all reports, and I havn´t found any really critical reports about JAIN or Parlay. This makes you wonder if the technology has the potential the authors claim and if the market will adopt the technology. I find the technology very interesting but I would like to see some comparisons with other technologies in the domain.

I got access to a development guide and javadoc from a company that develops a parlay gateway, and an application server for JAIN SPA. Even though I got access to this documentation I had trouble to understand how to develop applications for JAIN. I think examples, tools and testenvironments are cruisal for independent developers, and I hope more books, guides, and reports about development for Parlay and JAIN will be published in the future.

8.2.2 Test environments

To test a JAIN application can be as hard as to develop it. You need a lot of software and hardware to test this kind of applications. To be able to fully test your application you need either a real or simulated telecommunication network, a Parlay gateway, and probably also an application server that provides access through the JAIN SPA. This is all expensive equipment and not anything you can get without investing a large amount of money or partner up with a network operator.

Developing applications for the telecom domain might be a lot easier than before with JAIN and Parlay technology, but will neither way be anything you can do without having access to expensive testing equipment.

8.3 Summarizing my experiences with JAIN

All the functionality I needed for the VPBX is provided by Parlay and JAIN. It is a rich framework with lots of functionality. Besides multiparty call control, JAIN provides functions for checking user status on mobile phones, functions for interaction with users through dial tones and voice, positioning of mobile phones, messaging (SMS) and content based charging. This is enough functionality to create advanced telecom applications for the Parlay and JAIN framework.

I had only a brief understanding of telecommunications when I started working on this thesis. Therefore I have had a rather hard time understanding Parlay, JAIN, and the telecommunication networks. However my knowledge of this domain has increased and I now have a good understanding of how the telecommunication networks and services link together. As said before I think much more should be written about the subject. To get developers to create applications for the JAIN and Parlay framework, more tools, examples, books and other resources such as developer communities on the Internet needs to exist. I had a hard time figuring out how to use the framework, without this resource.

Even though I have had trouble with understanding how to use the framework, it really is useful. I believe that many more developers will start creating applications that use the telecom networks if Parlay and JAIN will be common and used by most network providers. With Parlay and JAIN technology, costs and time for developing applications that use the telecom network will greatly decrease. With the mobile phones of the future, with large memory and displays, you can probably use Parlay and JAIN applications on the client side. The possibilities are enormous.

9 FUTURE WORK

A lot of future work is needed in order to find out how to create the VPBX. To answer the question if Parlay/JAIN is a good technology for such a system, a prototype needs to be developed and evaluated. Therefore the most important future work is to create a more detailed design of the VPBX and a test implementation of this design. After that the system needs to run in a simulated or real telecom network, to be evaluated. The system also needs to be compared with other PBX solutions on the market.

To be able to run more than one company on one VPBX, the system needs probably to be distributed. One or several modules could run on different hardware. A future work could be to analyse and test how to scale up the system to enable several companies to run on the same VPBX. Distribution might be one way, but other approaches probably exist. How should the design be changed to support distributed modules? How does it affect the performance of the system if it is made as a distributed system?

Also running more than one similar module in parallel to increase the performance and/or availability relates to the distribution of the VPBX.

A very interesting work could be to analyse how the system can interact with a network operator. Many features and services is already enabled and provided by the network operator. How can the VPBX interact with these services in the best possible way? One such service is the voicemail system that all network operators nowadays provide as a service for their customers. Can this voicemail system be integrated with the VPBX system and accessible from the system?

There is probably many more features and services that the network provider owns that can be useful when improving the VPBX.

Finally a more thorough analysis of Parlay/JAIN would be very interesting. How does Parlay/JAIN compare to other call models for example? There are a several call models that exist today and others that are developed in parallel with Parlay and JAIN. Analysing these call-models and comparing them with each other would be of great interest. What are the strengths and weaknesses of each call model?

To be able to answer that question, a technique for comparison also needs to be developed.

10 CONCLUSION

The JAIN framework provides many advantages when developing software for telephony. It encapsulates the underlying signaling layer in the telecom network and provides a single interface to applications, whether the call is transported on an IP, wireless or PSTN network. JAIN handles all communication with the different services in the telecom network and provides a high level Java based API to application developers. It opens up the possibility for third party developers to create new interesting and useful services, both fast and without the deeper knowledge of low level signaling protocols. The Parlay and JAIN framework provides the functionality needed for a Virtual PBX for mobile telephony. It provides the possibilities to handle calls, check status on mobile phones, and interact with the users (callers) that this system needs. Besides that Parlay and JAIN also provides functions for mobility and charging. Economic aspects are out of the scope of this project, but are of course very important when creating such a product as the VPBX.

In the beginning of this report I presented three problem statements. The first was "How to design a virtual PBX system for mobile telephony?" I believe I have met that statement with the design solution presented earlier. The design did not become as detailed and complete as I wanted from the beginning, but this was not possible due to the time scope of this project and the complexity of the system. I misjudged the complexness and size of a software based system like this. Although I could come up with a design proposal on a higher architectural level, that provides a good platform and starting point for future work on the VPBX system.

The second statement was "Can it be done with the Parlay and JAIN technology?" By studying and trying out the JAIN framework (i.e. the JAIN Service Provider API) throughout this project I believe that the system can be created and run on JAIN/Parlay.

The third statement was "Is Parlay and JAIN a good technology for this type of product?" That is harder to answer since I haven't been able to implement and test a prototype of the system. I think a software based PBX based on JAIN can work, but if it is a good technology is not possible to answer. A prototype must first be created and evaluated before that question can be answered.

My experience of using the JAIN framework is mostly positive. The JAIN Service Provider API is a well defined Java API that is rather easy to understand. It provides all the functionality that most application developers will need when developing software for the telecom domain. Hopefully Parlay and JAIN will also be so network provider independent as the standard and its supporters say. This and the network convergence that Parlay and JAIN provides will, I think, provide a great platform for developers when creating new applications and services.

One thing I lacked though was good literature about Parlay and JAIN. I only found documentation about the standards and more general presentations of Parlay and JAIN. No examples or guides about how to develop software that uses the JAIN framework. This was the thing I missed the most when working with the JAIN framework and something I hope will change in the future, if JAIN and Parlay gains more market shares.

REFERENCES

Beddus S., Bruce G. and Davis S. (2000), 'Opening up Networks with JAIN Parlay', *IEEE Communications Magazine*, April, pp. 136-143.

Dictionary of Computing, Third edition (1998), Peter Collins Publishing, Middlesex.

Green J.H. (2000), *The Irwing Handbook of Telecommunications*, fourth edition, McGraw-Hill, New York

Jain R., Anjum F.M., Missier P. and Shastry S. (2000), 'Java Call Control, Coordination, and Transactions', *IEEE Communications Magazine*, January, pp. 108-114.

Jepsen T.C. (Editor), Anjum F., Bhat R.R., Jain R., Sharma A. and Tait D. (2001) *Java in Telecommunications – Solutions for Next Generation Networks*, John Wiley & Sons, West Sussex.

Keijzer J., Tait D. and Goedman R. (2000), 'JAIN: A New Approach to Services in Communication Networks', *IEEE Communications Magazine*, January, pp. 94-99.

Olsson A. (Editor) (1997) *Understanding Telecommunications 1*, Studentlitteratur, Lund, Sweden.

Olsson A. (Editor) (1998) *Understanding Telecommunications 2*, Studentlitteratur, Lund, Sweden.

Sun Microsystems Ltd (May 2002), 'The JAIN APIs: Integrated Networks for the Java Platform', white paper from Sun located at: www.java.sun.com/products/jain)

Tait D. (2001), 'JAIN APIs for call control and wireless networks', *Telecommunications development Asia-Pacific*, September, pp. 79-87.

INTERNET LINKS

All listed Internet links are reachable at present date: 9th of March 2003.

JAIN, Java API for Integrated Networks
http://www.java.sun.com/products/jain/

JCP, Java Community Process
http://jcp.org

The Parlay group
http://www.parlay.org

APPENDIX A: GLOSSARY OF TERMS

AEG	Application Expert Group, expert group in JCP
AIN	Advanced Intelligent Network
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BSC	Base Station Controller
BTS	Base Transceiver Station
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
GMSC	Gateway MSC
GSM	Global System for Mobile communication
H.323	ITU-T standard for packet based multimedia communications systems.
HLR	Home Location Register
IN	Intelligent Networking
INAP	Intelligent Network Application Part
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISUP	Integrated Services Digital Network User Part
JAIN	Java APIs for Integrated Networks
JAIN SCE	JAIN Service Creation Environment
JAIN SLEE	JAIN Service Logic Execution Environment
JAIN SPA	JAIN Service Provider API
JCC	Java Call Control
JCP	Java Community Process
JTAPI	Java Telephony API
MAP	Mobile Application Part
Megaco	Protocol used between elements of a physically decomposed multimedia gateway, defined in FRC2015.
MGCP	Media Gateway Control Protocol
MS	Mobile station, often mobile phone
MSC	Mobile Switching Centre
NMT	Nordic Mobile Telephony
PBX	Private Branch Exchange
PEG	Protocols Expert Group, expert group in JCP
PLMN	Public Land Mobile Network
PSTN	Public Switched Telecommunication Network.
SIP	Session Initiation Protocol
SMS	Short Messaging Service
TAPI	Telephony API
TCAP	Transaction Capabilities Application Part
VLR	Visitor Location Register
VPBX	Virtual PBX