NTNU

Norwegian University of
Science and Technology

Mobile Remote LAN

Designing a modular service platform

Lars Are Aschim
Lars Martinsen

Master of Science in Communication Technology

Submission date:	June 2009
Supervisor:	Van Thanh Do, ITEM
Co-supervisor:	Simone Lupetti, Telenor R&I

Norwegian University of Science and Technology
Department of Telematics

Problem Description

It is not uncommon for today's users to have access to more than one device, including PC's, PDA's and mobile phones. If the user wants to access services in his home network from a remote location, he has to manually customize a connection to each of these services. The purpose of this thesis is to make these kinds of services available without complex configuration.

This Master Thesis aims at providing a solution for accessing generic services on a user's home network from a remote device such as a PC, a PDA, or a mobile phone. The home network in this context can be any place where the system requests access to a service, such as in a user's home, office, car, cabin, etc. The system must allow access and use services from a wide range of devices and platforms. Such a solution should work as a service platform allowing the user to easily customize a set of services. The goal is to offer a solution to select, deploy and use generic home services from a remote device, such as PC's, PDA's or mobile phones.

As an example of a service we will focus on remote network search, i.e. the possibility of a remote user to search and retrieve information from his devices attached to the service platform while at a remote location. This example service is important because it deals with problems related to topology, expandability, and usability. In addition, it is also complex enough to highlight unforeseen issues with the service platform.

The final goal is to build a working service platform, offering remote network search based on a modular platform as an example service.

Solutions to non trivial challenges such as NAT and firewall traversal, bandwidth restrictions, modular system architecture, and usability are performed throughout the report to achieve the goal.

Assignment given: 15. January 2009
Supervisor: Van Thanh Do, ITEM

Acknowledgements

The project proposal was given by Simone Lupetti and Telenor R&I, and the problem definition was defined in co-operation with Simone Lupetti. The work has been executed at NTNU in Trondheim, but there have been regular meetings between the authors of this thesis and Simone Lupetti, both on phone and at Telenor at Fornebu. The duration of this specialization project was from 15.01.2009 to 11.06.2009.

This report is the product of almost five months of reading and researching. The report is written for Simone Lupetti and Telenor, but should also give an insight to problems and solutions to these kinds of systems to anyone interested.

A thank to our supervisor Simone Lupetti, who has guided us with useful comments and support regarding technical aspects and guidance. He has also given feedback throughout the whole process that has improved this Master Thesis.

A special thank also goes to our girlfriends for the understanding and patience at times when working long nights and early mornings, and for proofreading this report.

______________ ______________

Lars Martinsen Lars Are Aschim

Abstract

It is not uncommon for today's users to have access to more than one device, including PC's, PDA's and mobile phones. If the user wants to access services from a remote location, he has to manually customize a connection to each of these services. This thesis aims to make these kinds of services available without complex configuration, using a modular framework. As an example, new hardware or software might be needed in order to integrate home services and mobile devices. These circumstances make it hard for a regular user to deploy new services at home. At the same time people become more and more mobile, and users are moving from being passive consumers to interactive participants of the Internet.

The general idea of ubiquitous communication between hosts in the Internet is brought down to a practical level by creating a use case where a user would like to search and retrieve files present in his home network while at a remote location. Solutions to non trivial challenges such as NAT and firewall implications, bandwidth restrictions, modular system architecture, and usability are examined to make a modular service platform meeting the demands of more interactive and mobile environments. The XMPP protocol, mostly known as an instant messaging and presence protocol, is utilized to create a web of trust between services and users.

To achieve the goal of making a modular service platform, providing connectivity that allows services and users to be mobile, a working prototype has been made. The prototype consists of a modular service platform, enabling services to be added as plug-ins. The service platform is divided in two parts; one part enabling connectivity using a third party solution, and one part enabling a modular framework to add services as plug-ins. The functionality was tested with a network search as an example service, developed as a plug-in using the modular service platform.

Table of Contents

Figures

Tables

Abbreviations

AJAX	- Asynchronous JavaScript and XML
AmI	- Ambient Intelligence
AMZN	- Amazon
API	- Application Programming Interface
ASP	- Active Server Pages
CPIM	- Common Presence and Instant Messaging
CPU	- Central Processing Unit
CSS	- Cascading Style Sheets
DDOS	- Distributed Denial-Of-Service
DOM	- Document Object Model
EC2	- Elastic Compute Cloud
FTP	- File Transfer Protocol
GUI	- Graphical User Interface
HS	- Home Server
HTML	- Hypertext Markup Language
HTTP	- Hypertext Transfer Protocol
ICMP	- Internet Control Message Protocol
ICT	- Information and Communication Technologies
ID	- Identity
IDE	- Integrated Development Environment
IE	- Internet Explorer
IETF	- Internet engineering Task Force
IM	- Instant Messaging
IMPP	- Instant Messaging and Presence Protocol
IP	- Internet Protocol
ISO	- International Organization for Standardization
J2EE	- Java 2 Enterprise Edition
J2SE	- Java 2 Standard Edition
JID	- Jabber ID
JSP	- Java Server Pages
LAN	- Local Area Network
LGPL	- Lesser General Public License
NAT	- Network Address Translation
NS	- Network Server
OSI	- Open Systems Interconnection
PC	- Personal Computer
PDA	- Personal Digital Assistant
PDF	- Portable Document Format
POJO	- Plain old Java Object
POP	- Post Office Protocol
RCP	- Rich Client Platform
RFC	- Request For Comments
RFID	- Radio-frequency Identification
RIA	- Rich Internet Applications
S3	- Simple Storage Service
SASL	- Simple Authentication and Security Layer
SMTP	- Simple Mail Transfer Protocol
SOA	- Service-Oriented Architecture
SOHO	- Small Office Home Office
SVG	- Scalable Vector Graphics
TCP	- Transmission Control Protocol
TLS	- Transport Layer Security
UI	- User Interface
UTF	- Unicode Transformation Format
VM	- Virtual Machine
VPN	- Virtual Private Network
WURFL	- Wireless Universal Resource File
XEP	- XMPP Extension Protocol
XHTML	- Extensible Hypertext Markup Language
XML	- eXtensible Markup Language
XMPP	- eXtensible Messaging and Presence Protocol
XSF	- XMPP Standards Foundation
XSLT	- Extensible Stylesheet Language Transformations

Mobile Remote LAN Master Thesis

Part I - Introduction

Part one describes the evolution of the Internet today, and gives a context to this thesis. Different use cases describe how such a system can solve and ease problems such as interoperability, modularity, user friendliness, and connectivity of Internet based systems. This part also describes the motivation and problems related to the thesis, and give the reader a description of the main goal of the thesis.

Part one also gives an insight of related research projects and similar commercial systems, and show example systems and technologies with adjacent functionality.

1 Part I

1.1 Introduction

This chapter provides a context in which the service platform, created during this thesis, acts as an enabler for different types of services.

The Web 2.0 revolution, where users are moving from being passive consumers to being active publishers of their own content, has massive implications on how business is conducted on the Internet. It is possible to argue that the Internet is now moving from being an extension of the mass media regime of the twentieth century and into an Internet where consumers make the trends and decide when and where to turn next. Tim O'Reilley claims that the Web-consumer model is outdated; people are producers of content [1]. This development, fueled by emerging technologies which enable easy publishing of content to the web, e.g. in the form of blogs and wikis, is what the future web looks like [2].

At the same time as the consumer model is changing, the ways of connecting to the Internet have shown an exponential growth during the last couple of years. The number of uses of the Internet is growing at the same rate [3]. Internet access has moved from stationary, via spotty, through Wi-Fi access, to a situation where Internet connectivity is almost omnipresent through high speed mobile networks. People are used to accessing the Internet in the safe surroundings of their home, reading e-papers or writing e-mails to friends. In addition, people are now accessing the Internet while on the move, leaving short and sometimes informative bits of information for their friends and the world to follow, in near real time.

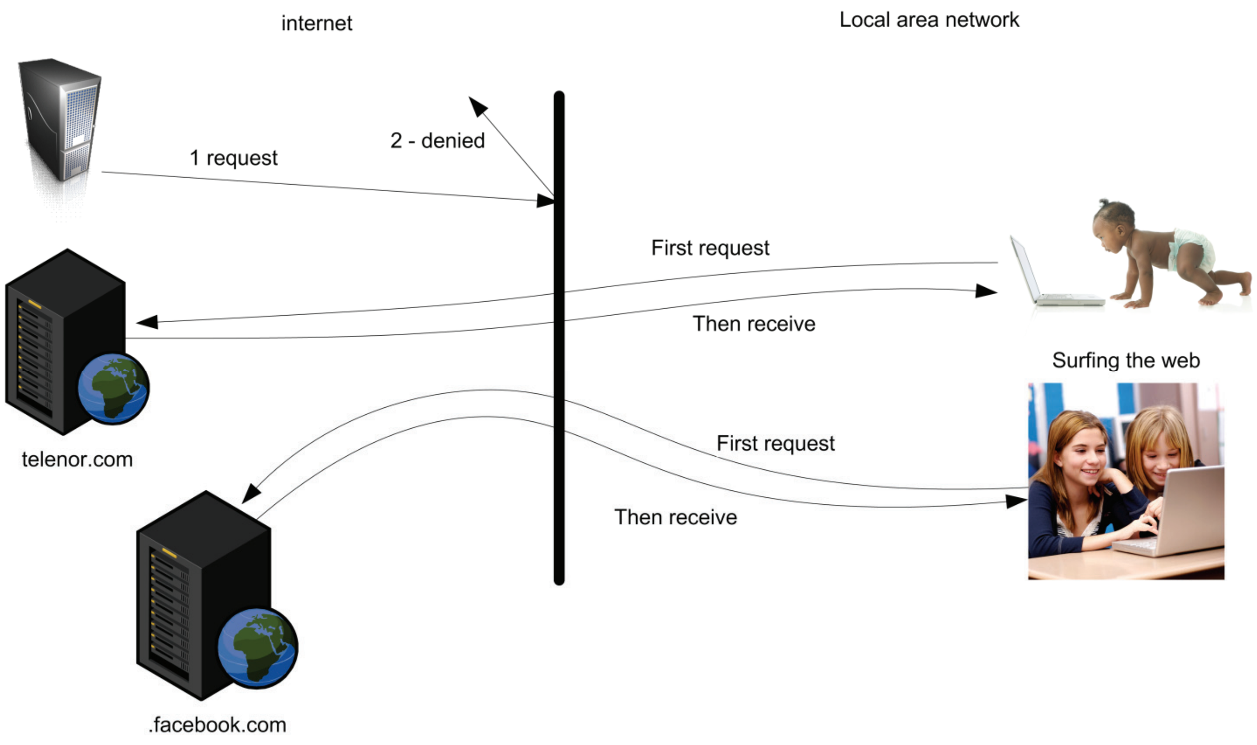


Figure 1 - Traditional Internet scenario

The scenario in Figure 1 shows how users are requesting information from the Internet, and get responses back. To enable this scenario, the user needs to know the address to the service he is requesting, and the service needs to know where to send information back, just like the postal system.

During the last fifteen years this scenario has been the prevailing use case on the Internet, however in the last couple of years this has changed. Users can access the Internet wherever they are, on their laptops, cell phones and other Internet enabled devices and accessing the same services they used from home; in addition they contribute more and more content themselves. Since the user is connecting to the same network from home and when traveling, it should be possible to connect from the remote location and back home directly. In this way it would be possible to retrieve documents, music, etc. from the stationary computer or to control any other device at home.

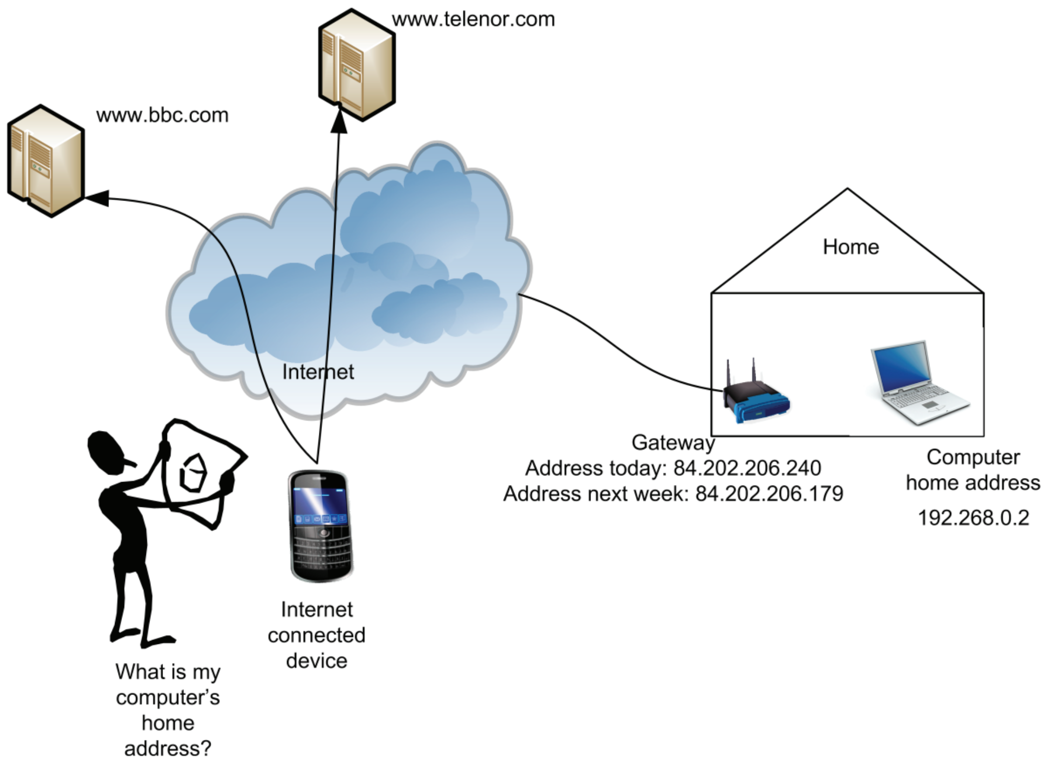

www.telenor.com
www.bbc.com
Home
Internet
Gateway
Address today: 84.202.206.240
Address next week: 84.202.206.179
Computer
home address
192.268.0.2
Internet connected device
What is my computer's home address?

Figure 2 - User trying to connect home

Figure 2 shows how a user is trying to figure out how to connect to his home computer. Given the scenario described in the previous paragraph, it should be simple; since all nodes connected to the Internet has an address, it should just be a matter of knowing the computer's home address. However, this is not the case, since the addresses allocated to users are dynamic. A dynamic address might change the next day or week, making persistent connections from the outside difficult. The addressing scheme has evolved this way for several reasons, the most important being the need of efficient distribution of addresses due to the lack of addresses. In addition gateways are used between the user and the Internet.

Due to the lack of addresses, it is common to get a single public address for all the computers in a home local area network (LAN). Generally speaking, when a computer behind a gateway access the Internet, a mapping is made between an internal address and an external address where traffic is relayed. This mapping is broken once the connection is terminated, and the outside cannot access the computer inside the LAN. There are workarounds to make persistent connections from the outside and in to computers behind these gateways, but such workarounds need configuration and is complex to setup for regular users.

In order to enable the development and integration of user-oriented services, this thesis looks at two main problem areas with today's Internet model.

- How to create a framework for ubiquitous communication in today's diverse Internet scenario
- How to make a modular service platform, to enable the integration and ease the deployment of new services in this context

The next paragraphs will explain these problem areas in some detail and explain the solutions proposed in this thesis.

1.1.1 Creating a framework for ubiquitous communication

A large part of the issue with today's Internet is the lack of an easy and persistent way of identifying a device in a home or office network. It is tied in behind a dynamic address and a gateway, which makes outbound connections easy. Inbound connections, on the other hand, are not allowed access. One solution is to establish a persistent connection between a computer at home and an outside host, e.g. a server located at Telenor as shown in Figure 3. The solution builds a framework on top of the dynamic addresses of home and office networks, and allows an outsider to connect back in.

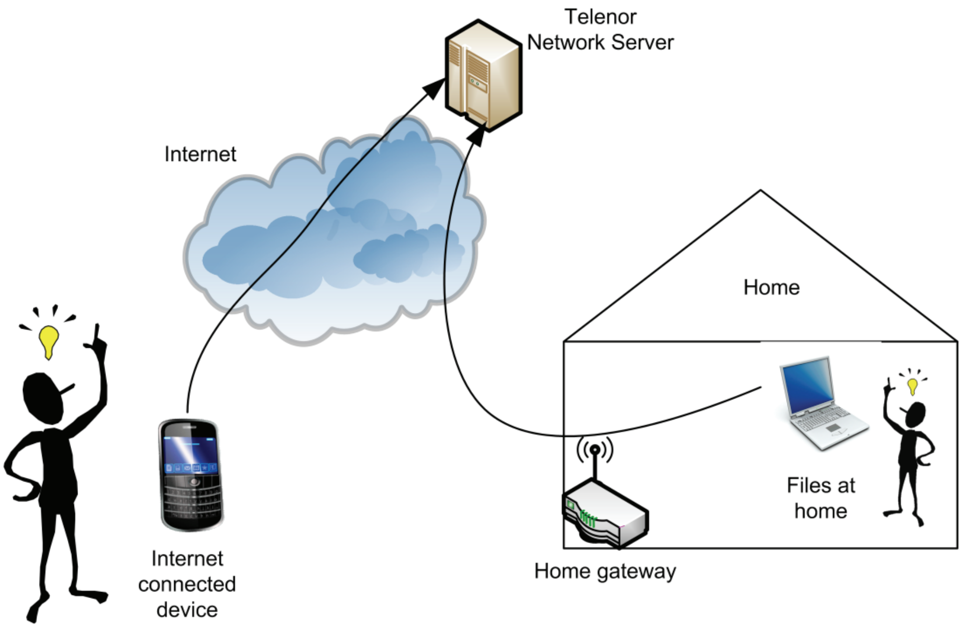


Figure 3 - Connecting to an intermediary host

Figure 3 shows how the user first connects the home and then the remote device to the Network Server at Telenor, which has an address easily remembered by users, e.g. *www.telenor.com*. When the users connect, they create accounts and log in, much in the same way users today access instant messenger (IM) or Facebook. IM protocols solve presence and connectivity problems using a third party server and enables users to connect to each other through this server. The account in an IM network might have the shape of an email address, e.g. *user@telenor.com* identifying the user and the server to which he is connected. The network server keeps track of all its users presence, e.g.

whether the user is online or offline and which users are friends with each other. Until now, the related context of this example has been a user accessing his home computer. This however, is only one scenario. The communication framework described here is designed to be universal. Its sole purpose is to provide devices with an easy way of communicating with users anywhere in the network, much like the original idea behind the Internet.

1.1.2 Modularity

The previous sections explained why inbound connections are denied and how the problem is solved using a third party and IM protocols. The connection is the backbone in the service platform created in this thesis; however the service platform must support services to be added dynamically to offer a modular platform to the users.

The IM protocol allows users to become friends with each other in a dynamic way. A person can add a friend to his roster, or list of friends, and get access to the friends' presence information and send messages to them. This feature is used in this Master Thesis to achieve modularity. The difference is that in this system, services are added to the roster instead of friends. Every device which is added to the service platform uses the IM protocol to offer its service to the user. Users add services to their roster to access the service. This solution creates a modular service platform, integrating services with the IM protocol.

Services without an Internet connection need an enabler to connect to the service platform, such an enabler can be a software application deployed on any hardware with Internet access and an interface to the service. The application will provide the communication to the service platform. Such an application is created, allowing services to be added as plug-ins to the platform described above.

By creating a modular framework with its own public Application Programming Interface (API), it will be possible to integrate services from other providers into the network, further increasing the value of the service platform. In addition, an advanced user might create and publish new service plug-ins. In the next paragraph an example of such a service is shown.

1.1.2.1 Remote Search

To create an interesting context and give an example of a service which might be added to the service platform, a specific service is developed during this thesis. This service was selected to identify and explain the complexity and issues surfacing when creating such a service platform. It is important to create a useful service which has the complexity needed to highlight issues in the service platform.

The service shows the modularity of the service platform, in addition to showing how a modular client might be installed and operated on an end user's computer. The service will implement a remote network search of the files on the user's computers and allow him to remotely search both the names and contents of his files residing at home. To complete the service, it is necessary first to display the files, and then add functionality to let the user download the requested file. This scenario is shown in Figure 4 where the user has an Internet connected mobile device and sends a request to the home network, while Figure 5 shows the response from the home network.

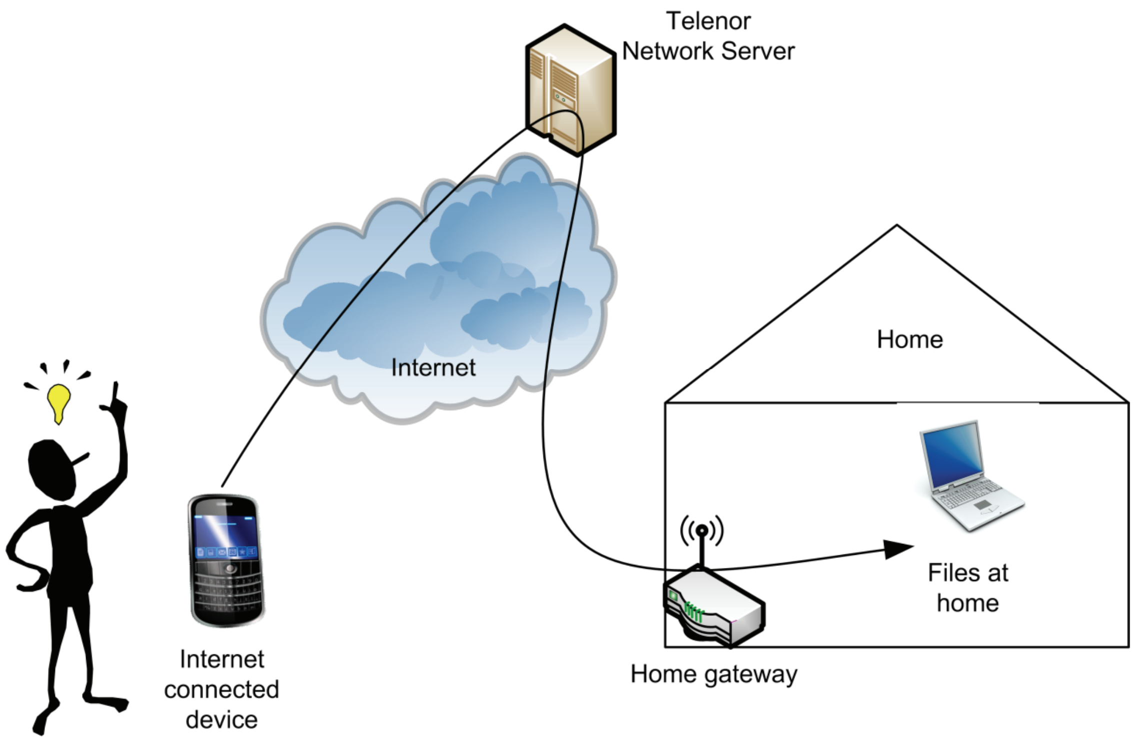


Figure 4 - Accessing files at home through Network Server

The purpose of this setup is to let the user, in possession of an Internet connected mobile device, access services available in his home network. In this example, the user wants to be able to access his home files, therefore he searches for the files through the network server. He receives a response back, containing a list of all the files which matched his search criteria, after which he can download the relevant files from his home to his Internet connected device.

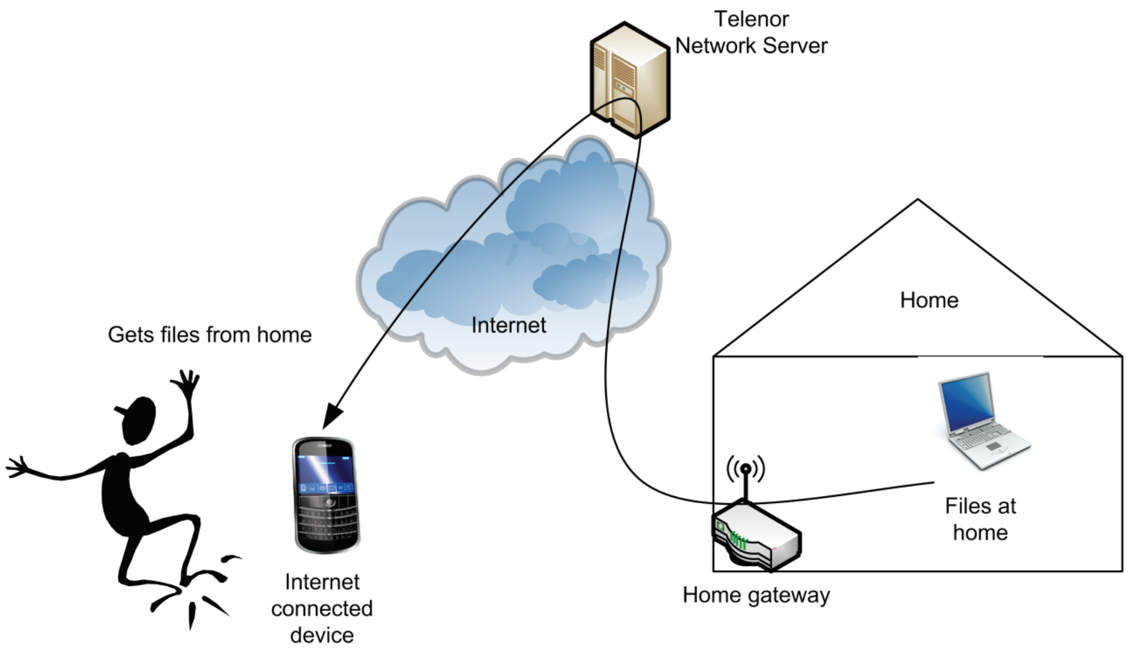


Figure 5 - Getting files from home

This scenario is a solution for generating a content search of the users' home files from a mobile platform. This case illustrates the different parts in the service platform. The user has an Internet connected mobile device, which he is using to access a service present on his home computer behind a gateway. This is achieved by accessing a web interface at a network server stationed somewhere on the Internet. There are many specifics of this setup which has not been discussed in this introduction; these are explained in Part III.

1.1.3 Use cases

As an extracurricular task performed while writing this thesis, a presentation of the thesis and the application was presented at the Telenor R&I department at Fornebu. The presentation forced the authors to take a step backward and review the accomplished work. In the preparation for the presentation the use case of an advanced home user was composed, and several others were discussed with the researchers at the R&I department. Some of these use cases are discussed in the next sections.

1.1.3.1 The advanced home user

The typical use case for a service platform like the one presented here, is the advanced home user, wanting to control and manipulate devices in his home, cabin, etc.

Services could include a power metering device, where the user would like to have real time monitoring of his power consumption. Equip the power metering device with an Internet connection; pre-configure it to connect to the network server, and experience real time monitoring of the device. The service platform is flexible and can be used to connect any Internet enabled device to the server or, as shown in the modular client in Figure 6, services can be connected to a modular client which provides connectivity.

Figure 6 is a sketch of how such a system might look. The components are structured as clients; some clients are devices with Internet access, while the client residing on the user's computer is enabled through a modular desktop application.

The specifics of the modular framework on the user's computer are described in more detail later on.

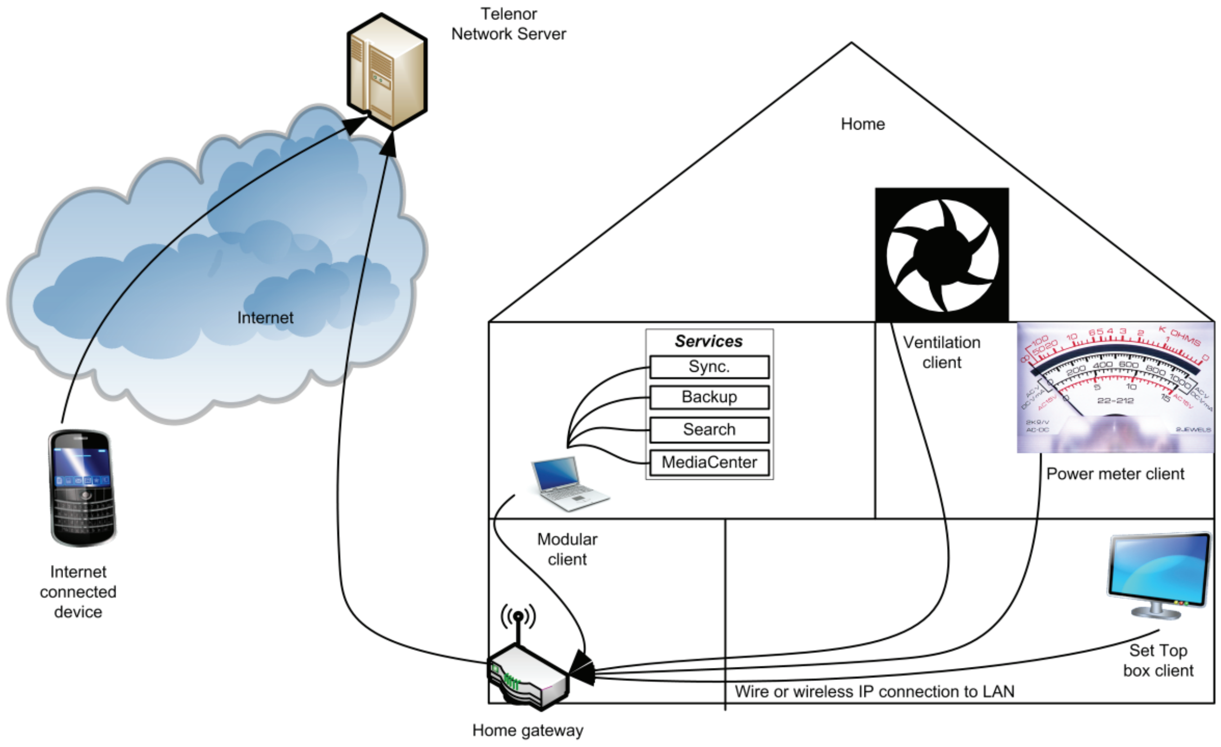


Figure 6 - The service platform and use case

1.1.3.2 *Public health*

Another interesting use case is to provide these services to companies or public health services. Today the Norwegian government uses massive amounts of money taking care of elderly people in retirement homes and institutions. These people require a substantial amount of tax payer money every extra day they have to stay in an institution rather than staying at home. An integrated communications system building on the proposed work of this thesis could incorporate sensors and alarms configured to increase the time elderly people could stay safe in their own homes. In this scenario, the cost/benefit ratio is quite different from the advanced home user. Today, health safety alarms already exist, but use proprietary solutions for communication. The integration of such a device into the communications framework proposed in this thesis could substantially lower the cost of such a deployment. At the same time this would enable the integration of sensors and devices in the system. With the opportunity to publish an open API, third party vendors would be able to contribute their own applications into the core system.

1.1.3.3 *Hospitals*

This kind of system would be equally interesting in a hospital environment, where integrated services are becoming a part of all aspects of daily life. In fact several systems already exist implementing the same protocol used as a foundation for the connectivity in this thesis. The XMPP protocol is an instant messaging (IM) and presence protocol [4] the hospitals use it to provide connectivity in large scale networks comprising thousands of clients [5].

1.1.3.4 *Tactical military networks*

The previous use cases have shown various civilian applications for the technology proposed. Another high cost application area for this kind of technology is in military tactical networks, where real time presence and location data should be transmitted reliably over unreliable and heterogeneous networks. Several implementations of this already exist today, and show the versatility of this kind of technology. The ability to negotiate the best path between any two network entities in near real-time is highly beneficial in a military setting [6].

1.2 Motivation

The main motivation for developing this kind of service platform is the idea of being free from location necessity and to provide usability to people with little or no technical background. The service platform should use personal devices as gateways to Information and Communication Technology (ICT) systems using the Internet as the communication channel. The platform should make it easy to plug-in new services at any time, ensuring the emergence of new services from third party developers and others using the platform.

Both people and devices become more and more mobile. The development of small powerful devices together with the increasing amount of devices available to consumers facilitates a more mobile ICT development [7]. To meet the demands of today, and prepare for the future, ICT systems must predict trends and make solutions for tomorrow. The authors believe mobility within ICT is one of the trends which will have a distinct effect on how people use ICT in everyday life.

People carry personal devices almost everywhere, and with the increase of data rates and coverage of cellular networks, this device has a potential of staying connected to the Internet at almost any time. The personal device is generally used as a phone and to send messages. In the last few years it has also become more and more common to use the device to send emails and browse the Internet, but the devices of today should be used more as gateways or as connection points towards other ICT systems and not just to browse the Internet or making phone calls.

The authors believe communication still is the main purpose of mobile devices, although not only towards other people, but towards ICT systems and services available in distant locations. Communication should not only be used towards popular Internet services of today such as Twitter, Facebook, Messenger, etc. but also include systems such as home- and car-alarms, identity (ID) recognition, healthcare, and other systems as defined in the Introduction. Also the "cloud computing" ideas of accessing information without thinking of the technology, backbone, or location are ideas the authors share for future ICT systems and technologies.

1.3 Problem description

It is not uncommon for today's users to have access to more than one device, including PC's, PDA's and mobile phones. If the user wants to access services in his home network from a remote location, he has to manually customize a connection to each of these services. The purpose of this thesis is to make these kinds of services available without complex configuration.

This Master Thesis aims at providing a solution for accessing generic services on a user's home network from a remote device such as a PC, a PDA, or a mobile phone. The home network in this context can be any place where the system requests access to a service, such as in a user's home, office, car, cabin, etc. The system must allow access and use services from a wide range of devices and platforms. Such a solution should work as a service platform allowing the user to easily customize a set of services. The goal is to offer a solution to select, deploy and use generic home services from a remote device, such as PC's, PDA's or mobile phones.

As an example of a service we will focus on remote network search, i.e. the possibility of a remote user to search and retrieve information from his devices attached to the service platform while at a remote location. This example service is important because it deals with problems related to topology, expandability, and usability. In addition, it is also complex enough to highlight unforeseen issues with the service platform.

The final goal is to build a working service platform, offering remote network search based on a modular platform as an example service.

Solutions to non trivial challenges such as NAT and firewall traversal, bandwidth restrictions, modular system architecture, and usability are performed throughout the report to achieve the goal.

1.4 Outline

This chapter gives the reader a short summary of the content in the report. The report is divided in four parts: Introduction, Theory, System Description, and Evaluation. For a complete chapter description the reader is requested to review the table of contents.

1.4.1 Part I - Introduction

Part one describes the evolution of the Internet today, and gives a context to this thesis. Different use cases describe how such a system can solve and ease problems such as interoperability, modularity, user friendliness, and connectivity of Internet based systems. This part also describes the motivation and problems related to the thesis, and give the reader a description of the main goal of the thesis.

Part one also gives an insight of related research projects and similar commercial systems, and show example systems and technologies with adjacent functionality.

1.4.2 Part II – Theory

This part gives an overview of the technical background, and a context to the technologies used to solve different problems related to this project. The different chapters provide insight into different technologies, and intend to highlight the main issues in this thesis; it does not aim to provide a complete understanding of the many technologies. Mostly standards and whitepapers from standardization organizations are referenced here and are recommended as further reading to give a more complete understanding of the technologies.

1.4.3 Part III – System description

Design choices used to realize the service platform and the example service, the remote network search, are discussed in the design chapter. This chapter focuses on discussing the motivation of choosing the different technologies. For technical details about protocols etc., please read part two. Part three gives the motivation of why a technology or software is used, but does not explain the technology or software in detail.

The second chapter of part three, the Implementation chapter, explains how the prototype is implemented. Code snippets, screenshots, installation procedures, and configuration details are described in this chapter.

1.4.4 Part IV – Evaluation

The testing, conclusion, and future work are revealed in part four. It explains the accomplishments of the thesis, and focus on future development to improve the service platform. Evaluation of cost is also revealed in this part in order to show the behavior of the service platform in a real life environment. Cost examples such as bandwidth and time consumptions, cost in NOK, and resource consumptions are shown. In addition a short reflection note is added to highlight the authors thought about this thesis.

1.4.5 Appendix and references

The last part of the report consists of references and appendixes. These documents explain system specifics in more detail, and have references in the report where it is expected.

The reference list uses the International Organization for Standardization (ISO) 690 - Numerical Reference standard of showing citations.

1.5 Related works

This chapter shows works and products with adjacent interests and technology. It is not a complete list of related works about similar commercial products or research projects, but it is an overview of the most interesting technologies and solutions in this context.

1.5.1 Cloud computing

Cloud computing is a wide term, but it is mostly used to describe web based services, and often used to describe a service available through a third party using Internet as a communication channel. "The cloud" store data away from the user, and hides the underlying technology while the user makes use of the cloud to process data. It should simply work on the device acquiring the service without extensive need for user configuration.

Cloud computing uses the Internet as the communication carrier, and therefore rely on web technologies to work. The cloud itself is built up to provide services from anywhere and from any device at any time. For these reasons, cloud computing must be built on a service-oriented architecture (SOA) and make use of automatic workload management [8] As Irving Wladawsky-Berger, Chairman Emeritus from the IBM Academy of Technology says:

"SOA is to cloud computing as HTML is to the Internet" [8]

One example of a "cloud service" is the web application DesktopTwo [9], which give access to a virtual desktop on a server located in "the cloud". This enables the possibility of accessing the desktop from anywhere at any time with the use of a web browser.

Cloud computing is related to this project when it comes to solving the issue of providing a service from anywhere and from any device at any time. The logic is located at a third party, in our case Telenor, offering the service in a cross platform manner.

1.5.1.1 Drawbacks to cloud computing

Cloud computing is not a communication service, but a term representing services put away from the user and processed by a third party, making it accessible from everywhere at anytime using a web browser. We want our system to be available from everywhere at anytime using a web browser, but the goal is not to put away services from the user.

In the case of our system, the third party is neither used to process the information nor is it the end point of connection. It is used to hide the complexity from the user and to achieve connectivity between a web client, using a web browser, and a home environment. Cloud computing can be compared with our system in the case where no information is stored on the mobile device, and all the complexity is hidden from the user. The main difference is that the third party is not an end point in our system and does not hold the services, the services are held by the home environment.

The problem related to this project is the interaction between the web client and the home network. The web client must have access and possibility to interact with the home network, therefore cloud computing is not an appropriate system solution. However, as a service running on top of our system it can i.e. be used to synchronization and backup of the user files on a home computer, storing the information away from the user.

1.5.2 Portable devices as data carriers

An optional system solution could be to use i.e. a mobile phone or Personal Digital Assistant (PDA) to hold all the information, and carry the information with you. This requires a complex system with interfaces to the entities to access in the home environment, and software developed especially for the mobile phone. A workaround for this could be to use a device developed for our system.

The main benefits of using portable devices with device specific applications as data carriers, are the access to multiple interfaces on the device such as: Radio-frequency identification (RFID), camera, Bluetooth, etc. These interfaces give the opportunity to communicate from the client using richer media as input.

1.5.2.1 Drawbacks to Portable devices as data carriers

If a mobile device is used to store information, the complexity raises in the context of the user because more logic must be in the portable device. A device will probably need a customized installation and configuration. This could be solved by using a preconfigured device developed specially for this system, but the cost will probably raise and a user must carry around another device to get connected to the service platform. This is not a good option, and it is better to include the service in a device that everyone already has.

1.5.3 Managed solutions

There is an ongoing research project at Telenor R&I investigating managed solutions. The next section is based on this research.

There are multipurpose Set-Top-Boxes, home gateways, and multimedia servers available which can be used in some context to the service platform. The managed solutions can work as a host for the software in the home network, and most of the logic can be in the device controlled by i.e. Telenor with less logic in the remote device open to the user. If a box is delivered to the user and no configuration is needed, the system platform is available to less technically experienced users and hence has a larger user base. A unified API should be available describing how interfaces must be created to access the system platform, making service development available to everyone and a wider range of services could be available to the users. This topic is under researching by Telenor, and the list below shows some available gateways by different manufacturers designed for home control.

- Homeseer [10]
- Xanboo [11]
- 4Home [12]
- Bulogics [13]
- Telsey [14]
- Homemanageables [15]
- BoBo Technologies [16]

There are also solutions where the HW can be small PC’s. Below is a list of some vendors offering these kinds of HW.

- Linutop [17]
- Aaeon [18]

1.5.4 Virtual Private Network (VPN)

One way to make secure connections to remote resources is to use VPN. VPN is a technology that virtually merges two or more private networks over non-private connections. In the context of this thesis, a virtual private connection between the mobile device and the LAN can be set up with an encrypted tunnel. This approach can prevent attacks on the channel, such as masquerade, replay, modification, etc.

VPN allows privileges on a remote device as if the device was connected inside a LANs firewall; it basically extends the LAN to incorporate the remote device as an entity with the same privileges as other entities in the LAN. There is a myriad of VPN technologies, these can be organized in two main categories; provider provisioned and customer provisioned VPNs. In the case of our system, customer provisioned VPN is the possible solution. In customer provisioned VPN the endpoints of the VPN tunnel exist of customer provided devices, and the tunnel itself has only one criteria; the packets have to be encapsulated within unique globally IP headers [19].

One example of a VPN service is the free and open source application OpenVPN. It can be used to achieve remote access in a secure way, using SSL. Open VPN has reached 3 million users and is ranked higher than any other solutions such as Checkpoint, Juniper, and F5 [20].

1.5.4.1 IPSec VPN

IPSec is one of the technologies in customer provisioned VPNs. It is a set of protocols that provides a secure communication through IP networks. Some of the features available by IPSec VPN are: encryption and integrity protection, and compression services performed by the IPComp protocol [21].

1.5.4.2 SSL VPN on mobile devices

> "SSL VPNs provide secure remote access to an organization's resources." … "The SSL VPN may also be accessed from any location …, as long as the location has connectivity to the Internet and the user has a Web client that is capable of using the particular SSL VPN." [22]

An application based VPN could be used in our system, but only as a security mechanism related to communication between the mobile phone and the LAN. SSL VPN provides e.g.; encryption and integrity protection, access control, endpoint security controls, and intrusion detection [22].

1.5.4.3 Drawbacks to VPN

There are several reasons why VPN is not used in our system. The three main drawbacks are: Complexity, reduced mobility, and additional overhead. Regarding the complexity, mobile phones must often be customized to support VPN, meaning VPN is not platform independent [22].

> "Some SSL VPN devices support phones via a browser, for web-based access, but few support them from an agent perspective, allowing access to client/server applications.“ [22]

The limited support to mobile devices and complex configurations are the main concerns of using VPN as a system solution. In addition, it does not address the problems related to modularity, but only connectivity.

Mobility problems related to roaming can also create problems to the connectivity. Any solution that relies on higher-level encryption has the potential to break when the user is roaming. There are some third-party solutions that address this shortcoming; however these solutions are not explained any further in this report. [23]

Due to the complexity and reduced mobility, VPN is not suited to solve connectivity problems between a remote device and a home network. In the next sections, commercial and research projects are examined to give alternative solutions to the connectivity problems.

1.5.5 Commercial products

Several commercial products exist implementing parts of the solutions described in this thesis. Some of these products are described in the following sections.

1.5.5.1 Soonr

Soonr [24] is a company and service provider giving users the ability to synchronize and access their documents from a web browser. Soonr uses a third party to store and backup files from the home network, where they can be accessed by the user and his friends. There is a small desktop client installed on a computer in the home network. The local files are copied to the third party server and are synchronized with the original files.

The architecture is similar to our system, a client is installed on a desktop computer, a web client is used to get access, and a third party is used to get connectivity. However, this is a single service offering backup and synchronization of files, not a communication framework to allow communication between the web client and the desktop application. This is where it differs from our solution. The web client and desktop application never communicate with each other, the web client only access the files on the third party and the desktop application only make sure that the files on the third party are synchronized.

The system also has a similarity with the example service developed in this thesis. It provides ubiquitous access to files from anywhere as long as the user has access to a web browser and an Internet connection.

1.5.5.2 Meebo

Meebo [25] is an Instant Messenger (IM) web client using the XMPP protocol to achieve connectivity between the web clients. This project uses XMPP on top of Hypertext Transfer Protocol (HTTP) as a communication protocol, and solves file transfer between two web clients using XMPP. Meebo has the ability to transfer files between two web clients, using JavaScript and XMPP as core technologies. This relate to the search service in our project, where downloading of files is an important aspect of the service.

Meebo makes use of Amazon (AMZN) Elastic Compute Cloud (EC2) [26] and AMZN Simple Storage Service (S3) [27] which are core features of AMZN Web Services, to have functionality for file transfer. The same approach could be performed in our project, but instead of using AMZN web services the third party in our system could be used similarly.

1.5.5.3 ISODE and XMPP for commercial, Government, Aviation and Military use

ISODE is a British company delivering communications solutions for a wide range of solutions in several commercial, government, aviation and military applications. They use XMPP to create closed collaboration solutions in all these applications. Some examples include using XMPP over radio and satellite networks for multi user chat sessions, white boarding, and publishing and sharing data with the help of XMPP's PubSub extension and setting up peer to peer communication such as; file transfer, voice and streaming video. ISODE has published several whitepapers describing their various use cases for XMPP which might be a good reference for understanding the XMPP technology. [28]

1.5.5.4 Eclipse RCP applications

The Eclipse RCP platform used in this thesis has been used in a number of both commercial and open source applications, some of which will be mentioned here. The interested reader should visit their web sites to get a more thorough understanding of the versatility of the Eclipse platform.

Eclipse RCP MP3 manager

The Eclipse RCP MP3 manager is a feature application used to show some of the features of Eclipse RCP, updated to show the features of the latest version of Eclipse. [29]

Azureus – BitTorrent client

Azureus [30] is an open source BitTorrent client based on the Eclipse Standard Widget Toolkit. It is a fully featured client, amongst the features are; plug-in based and extensible, three different levels of user interface, support for more than 40 different languages and a highly customizable user interface.

IBM Lotus Expeditor

IBM Lotus Expeditor [31] is one of several applications in the IBM Lotus series which uses Eclipse RCP as the framework for the application. Expeditor is a SOA client integration platform made to help developers integrate a wide variety of clients (RCP, Java, Web, AJAX and .NET) with the servers(Java EE, Web Services, Portlets and others) to reduce latency and workload and improve user satisfaction.

1.5.6 Research projects

The area of study covered by this thesis is also covered by several other research projects. Research projects in this genre are most often connected to a specific use case and might prove a good starting point for someone new to the subject. One large area of study when it comes to providing connectivity across diverse networks is telemedicine. More and more hospitals and health care providers are connecting all the devices and employees in their system to an online collaboration system. With the security and diversity of a large hospital in mind, the requirements for such a system are quite high.

1.5.6.1 XMPP based Health Care Integrated Ambient Systems Middleware

Labidi, Wael, et al. proposes using an XMPP framework as a middleware solution for connecting medical sensor networks across multiple domains [32]. They argue that XMPP can serve as a robust event-notification middleware solution to enable what they call Ambient Intelligence (AmI) where the emphasis is on user friendliness, distributed services support, user empowerment and support for human interaction. They propose a solution for using XMPP and specifically the XEP-0060 "PubSub" publish subscribe extension in a real-time large-scale Integrated Ambient system across health care domains. They also mention several related projects in the same health care context showing how this is an evolving field where several actors have started the development of collaborative solutions. The goal of the prototype developed in the paper about XMPP healthcare is to enable physicians to monitor the sensors connected to their patients in real-time.

1.5.6.2 The reliability of XMPP for file transfer

An employee of the "Helse nord IKT" carried out a project about the reliability of XMPP file transfer during his Master Thesis at the University of Tromsø. The organization is responsible for the ICT infrastructure in and between the health care providers in the northern part of Norway. The study investigates the possibility of using XMPP and its extension for file transfer to enable the transfer of large files, e.g. x-ray photos in a more efficient way than the Post Office Protocol (POP)/ Simple Mail Transfer Protocol (SMTP) solution used today. These large files currently create a bottleneck in the network as they are mostly transferred by e-mail between institutions and are stored at mail servers until the recipient downloads the file. By using the SOCKS5 file transfer, the Master Thesis shows how XMPP might improve this situation and contains availability and performance measurements [33].

1.5.6.3 The ISIS project

The ISIS project -"Infrastructure for Integrated Services" is a research project conducted as collaboration between NTNU, HiA, Tellu, Ericsson and Telenor to create a new integrated approach for services on diverse platforms. The project is a different approach to do some of the same tasks undertaken in this thesis. The project uses a tool suite called ARCTIS and a code generator called RAMSES to automatically generate code based on UML2 state machines. One of the focus areas of the project has been home automation. Recently they have focused on integrating SUN spots in home applications like alarm systems [34] [35] [36].

Mobile Remote LAN Master Thesis

Part II - Theory

This part gives an overview of the technical background, and a context to the technologies used to solve different problems related to this project. The different chapters provide insight into different technologies, and intend to highlight the main issues in this thesis; it does not aim to provide a complete understanding of the many technologies. Mostly standards and whitepapers from standardization organizations are recommended as further reading to give a more complete understanding of the technologies.

2 Part II – Theory

2.1 Firewalls

A firewall form a barrier through which the traffic going in each direction must pass. It can be implemented in either software or hardware, or both. It is the policy of the firewall that dictates witch traffic is to be authorized to pass in each direction. Firewalls can be implemented and used in many different ways, i.e. firewalls employed inside a Local Area Network (LAN) to restrict access to sensitive information or as the gateway out from the LAN and to the Internet. Firewalls can be categorized in three, and works on different layers in the Open Systems Interconnection (OSI)/ International Organization for Standardization (ISO) model:

- Network layer gateways
- Application layer gateways
- Circuit level gateways

2.1.1 Packet filter firewalls

Packet filter firewall is a network layer firewall, it generally base the policy on source address, destination address, and ports in each Internet Packet (IP). This is the most used firewall in Small Office Home Office (SOHO) networks, and it has two major strengths; it is quick and flexible. It is quick because it does not examine data above the network layer, and it is flexible because most modern network protocols can be accommodated using the network layer and below. This means that packet filter firewalls can be used to secure nearly any type of network communication or protocol. Packet filtering firewalls and routers can also filter network traffic based upon certain characteristics of that traffic, such as; filter out the Internet Control Message Protocol (ICMP).

Attackers have used this protocol to flood networks with traffic, thereby creating Distributed Denial-Of-Service (DDOS) attacks. Packet filter firewalls also have the capability to block other attacks that take advantage of weaknesses in the Transmission Control Protocol (TCP)/IP suite. This capability makes them ideal for placement at the outermost boundary of an untrusted network, as shown in Figure 7. At the boundary, the firewall can block certain attacks, filter unwanted protocols, and perform simple access control. By performing this basic filtering, it reduces the processing demand on other firewalls that examine layers above the network layer. [37] [38]

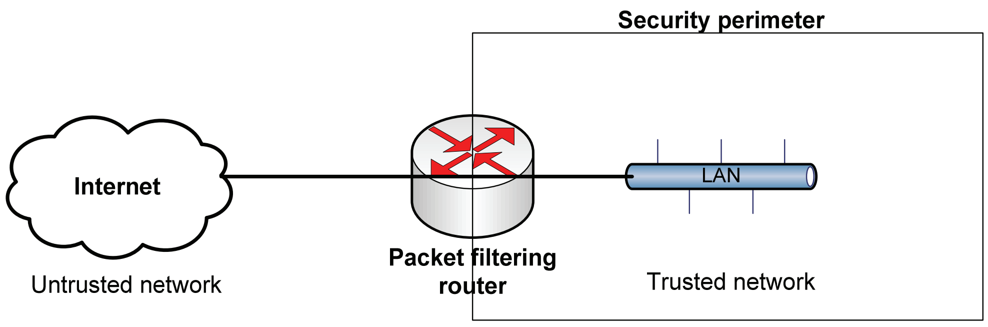


Figure 7 - Firewall, schematic of a packet filtering router

2.1.2 Application-proxy gateway firewalls

Application-proxy gateway firewalls or proxy-servers are application level gateways, and are more advanced firewalls than Packet filter firewall due to the combination of lower layer access control with application level functionality. Application-proxy gateway firewalls do not need the network layer functionality to rout; it routs the traffic between the inside and outside host based on software, as well as the examination of packets traversing the gateway is based on software installed on the proxy-servers. The policy/settings of the software determine if data packets are rejected or not. In addition to the policy, authentication can be performed differently:

- User ID and Password Authentication
- Hardware or Software Token Authentication
- Source Address Authentication
- Biometric Authentication

The authentication is one of the advantages of an application proxy gateway firewall. It is also easier to log and audit traffic passing the firewall. Proxy-servers can also be made less vulnerable to address spoofing attacks. There are also disadvantages using this type of firewall. Two disadvantages are addition processing overhead on each connection, and that the gateway must examine and forward all traffic in each direction. Because of this, this type of firewalls is not suited for high bandwidth or real time applications. Figure 8 illustrates the application level gateway. [37] [38]

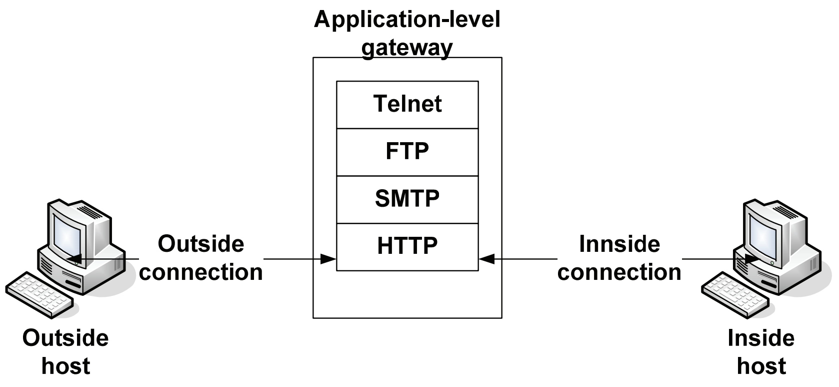


Figure 8 - Firewall, schematic of an application-level gateway

2.1.3 Circuit level gateway

Circuit level gateway can be a stand-alone system, or a specialized function performed by an application level gateway. The security functions consist of determining which of the connections that should allow access to the inside connection. When a request is received, the gateway checks the policy and then determines if the connection will be allowed or not. The policy is typically based on:

- Destination IP address and/or port
- Source IP address and/or port
- Requested application protocol
- Authentication
- Time restrictions

When the connection is set, a table is updated for the valid connections and the state information. All packets that are in the table will pass through the firewall without further validation. Because of this feature, a circuit level gateway is usually faster than an application level gateway. It is not an end-to-end connection, but the connection set up two TCP connections; one between itself and the inside host, and the second between itself and the outside host. The architecture is illustrated in Figure 9. [38] [39]

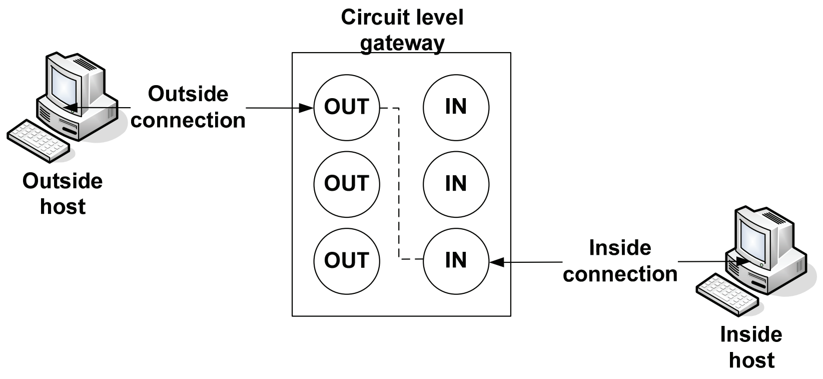


Figure 9 - Firewall, schematic of a circuit level gateway

The different types of firewalls have one feature in common, which is to block unwanted data packets going in or out through the network. This is also a disadvantage from this project point of view. If the services that we want to provide from inside the LAN and to a remote location are blocked; the system will not achieve connectivity. Therefore it is important that the firewalls do not block the data packets generated by our system.

2.2 Network Address Translation (NAT)

NAT [40] is sometimes thought of as a firewall technology, and NAT can have the same functionality, but it is a routing technology. NAT is used to translate the Internet traffic from outside the LAN, to different nodes inside a LAN due to the lack of IP-addresses. NAT solves this problem by mapping internal addresses to external or public addresses. An internal IP address: port pair is mapped to an external address: port pair, and whenever the NAT receives a packet to the external address: port pair, it knows how to reroute the packet back to the internal address: port pair. Applications with fixed port numbers are allowed to penetrate NAT enabled firewalls with static configuration. Applications with dynamic port numbers will be blocked with a NAT enabled firewall. The following sections are based on the paper: NAT traversal in SIP [41]. There are four types of NATs:

- Full Cone
- Restricted Cone
- Port Restricted Cone
- Symmetric

In a full cone NAT, everyone outside the LAN that knows the mapping schema can send packets to the client behind a NAT enabled firewall. This may be a security concern, and makes the clients susceptible to port scan attacks.

When a restricted cone NAT is enabled, no one from an outside connection can start sending packets to a client inside the NAT enabled firewall, before the client itself sends a packet in the outgoing direction.

As the restricted cone NAT, the port restricted NAT does not allow any outsiders to send the first packets to a client inside the NAT enabled firewall. In port restricted NAT both the IP address and the port number must match, not only the IP address as in restricted cone NAT.

In symmetric NAT, specific mapping of internal address: port pair to the NATs public address: port pair is dependent on the destination IP address that the packet is sent to. This is a common configuration in large enterprises.

2.3 Web technologies

This is an introduction to some of the web technologies used to realize this thesis. It will be an introduction and highlight the protocols in context with this project, and not an in-depth description of the protocols.

2.3.1 Asynchronous JavaScript and XML (AJAX)

AJAX [42] [43] is a collection of technologies, and not a single technology in its own. It is an abbreviation for Asynchronous JavaScript and XML, but AJAX is based on more technologies than just JavaScript and XML. At a minimum, JavaScript, XMLHttpRequest and the Document Object Model (DOM) are required for AJAX to function. In addition, XHTML and Cascading Style Sheets (CSS) should be used to handle dynamic display of a webpage. AJAX technology has been popular because it preserves and uses the web based infrastructure, while developers have the possibility to make rich Internet applications (RIAs). In the Web's next generation, referred as web 2.0, AJAX technologies are the key technologies.

The main goal of deploying AJAX is to make web based application more like desktop based applications [44]. This is achieved by the asynchronous data transfer used in AJAX, and not in a synchronous way such as traditional web applications. Operating asynchronously means that the browser can handle data whenever the data is sent from the server, running the logic in the background of the web page. Basically AJAX technology allows the user to communicate with the AJAX applications, while the AJAX engine communicates with the server when needed. In Figure 10 the concept of the AJAX web application model is illustrated together with a traditional web application.

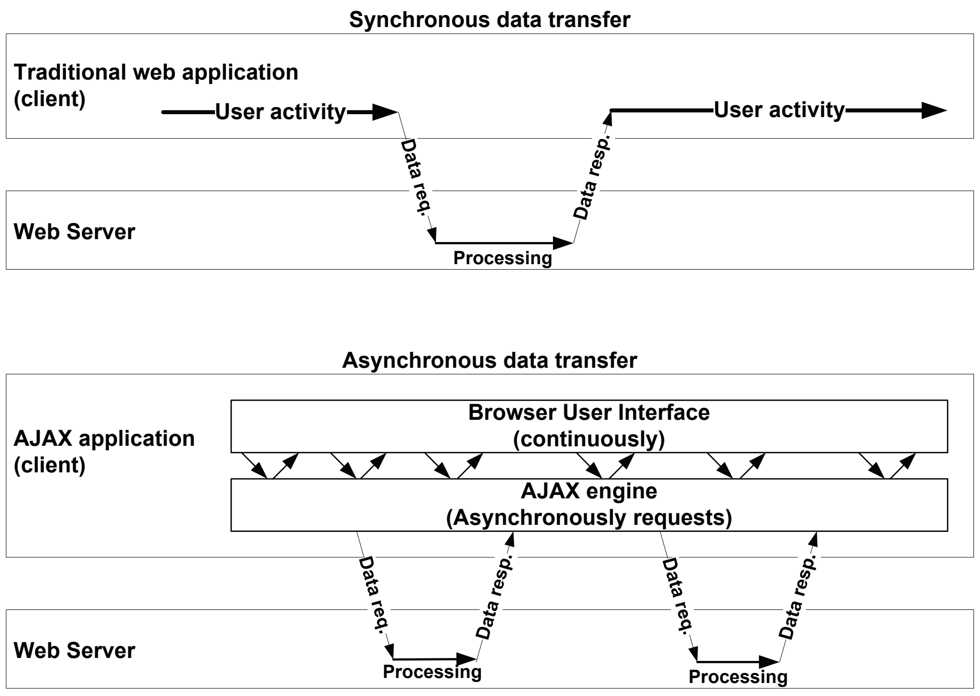

Synchronous data transfer
Traditional web application (client)
User activity
Data req.
Data resp.
User activity
Web Server
Processing
Asynchronous data transfer
AJAX application (client)
Browser User Interface (continuously)
AJAX engine (Asynchronously requests)
Data req.
Data resp.
Data req.
Data resp.
Web Server
Processing
Processing

Figure 10 - Concepts of synchronous and asynchronous browser behavior.

2.3.1.1 AJAX on mobile devices

Mobility becomes more and more important in everyday life, and AJAX has several approaches to cross-device application and mobility. Mobile AJAX is based on the same standard as AJAX, but extra considerations must be considered when developing a web page for both desktop and mobile devices. Screen size and bandwidth are two of the most critical aspects due to web applications on mobile devices.

The screen size can often be a problem when accessing a web page from a mobile device, one of the solutions in AJAX is to use CSS to provide both a desktop and mobile view of the web page while the content remains the same. One of the problems with CSS on mobile devices is that CSS is not implemented universally. There are alternatives to CSS, such as Extensible Stylesheet Language Transformations (XSLT), but the best solution may be to make a simpler page for mobile devices. The functionality should be the same, but the extensiveness and graphical presence should be left out when displayed to a mobile device. This approach requires more advanced server side coding, but will be more reliable and will most likely give the user a better experience.

Accessing Internet using cellular networks may result in unreliable connections, slow connections and more latency than in fixed networks. By limit the usage of images, external scripts and CSS files, can avoid long startup delay and latency when browsing the page. Asynchronous data transfer should be used widely to accept user inputs even though the device is offline. In addition, the use of asynchronous data transfer make a web application more attractive to mobile users since only parts of the page is refreshed from a server each time a user interacts with the application. When parts of the page are refreshed, the bandwidth consumption is lower, which result in lower costs for the user. If parts of a web page rely on network connection, the application should disable these parts when the connection is lost.

Issues such as battery, memory and Central Processing Unit (CPU) constraints, and also keyboard (or the lack of keyboard) should be taken into considerations when developing a web page for mobile devices.

AJAX also allows real-time applications such as chat and IM to be developed on web based applications on mobile devices, due to push technology in the web server. This enables a rich set of applications former restricted to desktop applications.

2.3.1.2 Same-origin

A problem occurring in web applications is the same-origin policy mechanism. The same-origin policy is a protection mechanism applying to HyperText Markup Language (HTML) documents preventing applications to interact over cross domain, cross protocol, or cross port. An AJAX proxy can be used to bypass this mechanism and accessing a third party server using XMLHttpRequest. The web server can be configured to bind the request to another port or domain in order to connect to other servers than the origin of the web page. The AJAX proxy can retrieve data on behalf of the client and relay the response.

2.3.2 JavaScript

JavaScript [42] [45] is what makes AJAX happen in the browser. More precisely, the XMLHttpRequest object inside JavaScript is what makes AJAX possible. It was developed from LiveScript by Netscape, before the name change in 1995. JavaScript is an object-oriented scripting language which is designed to be embedded in other applications such as web browsers. JavaScript only supports a few core set of objects, but it is extended with a variety of additional objects such as client-side JavaScript. Client-side JavaScript is used in web browsers to control various browser functionalities.

A standardization of the JavaScript language was performed in cooperation between Netscape's JavaScript, Microsoft's JScript and Ecma International [46], the European association for standardizing information and communication systems. The standardized version called ECMAScript behaves the same in all supported applications. In Table 1 the development of JavaScript and ECMAScript can be followed.

Table 1 - Overview of the JavaScript and ECMAscript standardization process [47]

JavaScript version	Relationship to ECMAScript edition
1.1	ECMA-262 [48], Edition 1 is based on JavaScript 1.1.
1.2	ECMA-262 was not complete when JavaScript 1.2 was released. JavaScript 1.2 is not fully compatible with ECMA-262, Edition 1, for the following reasons: • Netscape developed additional features in JavaScript 1.2 that were not considered for ECMA-262. • ECMA-262 adds two new features: internationalization using Unicode, and uniform behavior across all platforms. Several features of JavaScript 1.2, such as the Date object, were platform-dependent and used platform-specific behavior.
1.3	JavaScript 1.3 is fully compatible with ECMA-262, Edition 1. JavaScript 1.3 resolved the inconsistencies that JavaScript 1.2 had with ECMA-262, while keeping all the additional features of JavaScript 1.2 except == and !=, which were changed to conform with ECMA-262.
1.4	JavaScript 1.4 is fully compatible with ECMA-262, Edition 1. The third version of the ECMAScript specification was not finalized when JavaScript 1.4 was released.
1.5	JavaScript 1.5 is fully compatible with ECMA-262, Edition 3.

In the Previous section about AJAX, several of the approaches to make a web page user friendly and effective are implemented using JavaScript. JavaScript allows a browser to validate forms of data, dynamical content change, and push data from the web server. As explained, it is JavaScript that is the core technology in AJAX, making it all come together as a whole.

2.3.3 XMLHttpRequest

XMLHttpRequest [49] [50] provides full access to the HTTP protocol, and can return the web servers responses either synchronously or asynchronously, and both as text and DOM documents. It is not restricted to XML, but supports a text based format. It is neither restricted to the HTTP protocol, it also supports secure HTTP and other protocols.

When XMLHttpRequest is used to script HTTP, the process is:

- Creating a XMLHttpRequest object
- Specifying and submitting a HTTP request to a web server
- Retrieving the server response synchronously or asynchronously

The XMLHttpRequest object is not standardized and the process of creating the object is different in Internet Explorer (IE) and other browsers, but once it is created the use of the object is the same.

2.3.4 CSS

CSS is basically a text file separating the style and content of documents, making it easier and more efficient to maintain and develop web pages/web applications. AJAX relies on CSS to update and display results from e.g. the XMLHttpRequest object in the browser, enabling dynamic display. There are two standards covering CSS from W3C: CSS1 [51] and CSS2 [52]. CSS2 is based on CSS1 and have all the functionality of CSS1, meaning that a user agent supporting CSS2 understands CSS1. If a user agent only have support for CSS1, the structure of CSS will allow the user agent to read the document, but ignore the extended features of CSS2. W3C also have a recommendation for CSS on mobile devices which in general are a subset of CSS2 [53].

Using CSS decreases download time of a web page, i.e. it is less data to be transferred since all the style information only have to be coded once in a document and not for every element in a web page. This is an advantage when designing web pages to be displayed on both desktop computers and mobile devices, but there are more than download time CSS can do in order to increasing the user experience on mobile devices.

2.3.4.1 Mobile CSS

When a web page is displayed on a mobile device, it is an advantage to know what the user wants to perform on the web page. If the developer can predict the user desires and behavior, it is much easier to meet the needs of the user both quickly and simple –making the web page user friendly.

CSS allows formatting of documents to be displayed correctly on different types of media, some of the most important formats are:

- All, (used for all media types)
- Screen, (used for colored computer screens)
- Handheld, (used for small monochrome devices such as mobile phones and PDAs)
- Print, (used for print preview of a web page)

The handheld tag could be used to detect a mobile device, and support the content with a specialized display used for mobile phones and PDAs, but there are problems with this approach. Not all browsers support CSS or the handheld media type, and just display the content using the screen media type in its place. One solution could be to use JavaScript to detect the browser type using *navigator.useragent,* but not all browsers, at least mobile browsers, support JavaScript.

There are other solutions such as server side languages like PHP [54], Active Server Pages (ASP) [55], Java Server Pages (JSP) [56], etc., or another approach using an XML configuration file; Wireless Universal Resource File (WURFL) [57].

The server side languages are a cross platform solution, using the information about the user agent to detect device. The problem is that it is a huge quantum of user agents, and the list of user agents should be updated over time. WURFL is basically a XML file with information about user agents and what the user agents support. WURFL is an open-source and is updated regularly, making it an alternative to detect user agents.

By determine the user agent of a web browser; the web page can be configured to display the content to the user in the most preferred way, but it is not always a requirement to know the user agent. It may be enough to check if the device supports e.g. JavaScript and CSS. If it does not, it should display to the user an explanation that the user agent does not have support to view this page.

2.3.5 XML

XML [58] is a plain text language easily read by both humans and computers, and is used to transport and store data. The data in a XML document is wrapped in tags, where the tags are not predefined but they are case sensitive. The basic syntax of an XML document is that tags must be properly nested, and must have a root element.

Software that can process plain text can process XML, but XML-aware software can process the XML-tags specially.

2.3.6 XSLT

XSLT [59] is the recommended style sheet language of XML, and can be used to transform a XML document in either the browser or the web server. XSLT describes how the XML document should be displayed, and can transform a XML document into a new XML document, a HTML document, an XHTML document, etc. Browsers can transform the XML document differentially due to the XML parser. To ensure that the transformation is performed as intended, the transformation could be performed on the web server. XSLT consists of three pieces:

- XSLT, used to transform XML documents
- XPath, used to navigate in XML documents
- XSL or XSL-FO, used to format XML documents

XSLT uses XPath to look for a matching predefined template, if a match is found the document is transformed according to the template and creating a result document.

2.3.7 Mobile browsers

The gap between mobile and desktop browsers is decreasing, and mobile browsers become more powerful and in some cases use the same rendering engine as desktop browsers. One of the latest market researches [60] shows that Nokia have more than 40% of the world's total market of smart phones, followed by Research In Motion (BlackBerry), and IPhone. In the next few years more mobile browsers with full support for web 2.0 technologies will be available for users, both pre installed on the device, but also available for download on Internet.

In the next sections, different vendors and types of mobile browsers are examined to reveal support and functionality for web 2.0 technologies.

2.3.7.1 Nokia, S60

The web browser on S60 [61] is based on KDE's Konqueror open source project [62] using the Lesser General Public License (LGPL) components WebCore and JavaScriptCore, making the web browser open source with open APIs [63]. Figure 11 shows the components and the architecture of the browser.

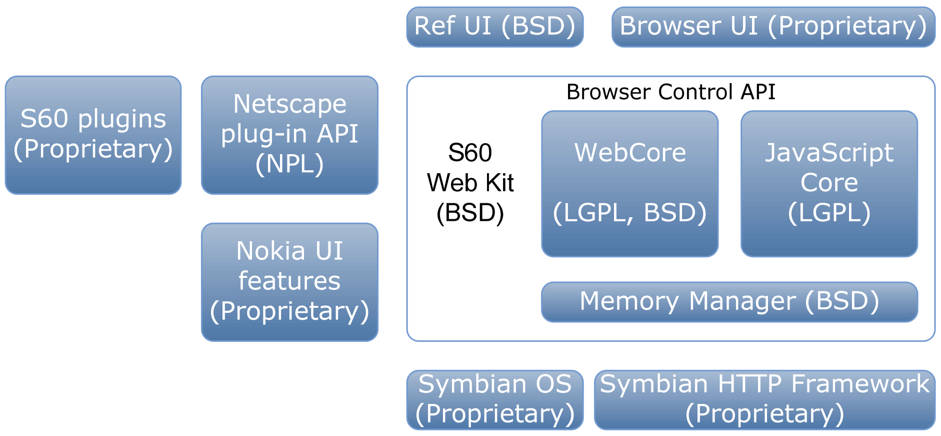


Figure 11 - S60 browser architecture

The browser has wide support for standards such as various W3C standards (HTML 4.01, XHTML 1.0, CSS 1, 2, 3 (partially), DOM 1, 2, Scalable Vector Graphics (SVG)-Tiny), ECMAScript, and Flash Lite. S60 is shipped with a number of devices from: Nokia, Samsung, LG, Lenovo, and Panasonic [64].

2.3.7.2 IPhone, Safari

IPhone [65] is the worlds' third largest smart phone distributor, but it is the most preferred phone to surf on Internet [66]. The mobile version of Safari is based on the same open source WebKit as the desktop browser, but does not support all the same CSS and JavaScript features as the desktop counterpart. The mobile Safari web browser is based on: HTML 4.01, XHTML 1.9, CSS 2.1 and partially CSS 3, ECMAScript 3 (supporting the XMLHTTPRequest JavaScript object) [67]. There are limitations to mobile Safari such as: XSLT support is limited to XSL page processing instructions embedded at the top of an XML page, input type='file', and Flash [68]. A new version of the IPhone OS is available at beta version [69], and will be available to download summer 09. OS 3.0 is predicted to be up to x10 times faster than the current stable release OS 2.2 [70].

2.3.7.3 Research In Motion, BlackBerry

There are two generations of BlackBerry browsers [71], first and second generation. The first generation browser only supports basic JavaScript and CSS. The second generation browser was introduced in version 4.6 and supports more advanced Internet technologies such as: HTML 4.01, CSS 2.1, DOM level 2, and JavaScript.

The three next browsers are not tied to any device, but are mobile browsers available to download and installation on a variety of devices.

2.3.7.4 Opera Mobile

Opera Mobile is installed on a variety of mobile phones, both preinstalled and available for download, and has since 2004 more than 120 million shipped installations [72]. Most smart phones can download the browser for a fee of approximately $20, but there are beta versions available to test the most recent releases. The last stable version, Opera Mobile 8.86, is available to both smart phones and pocket PCs with OS: Windows Mobile 2003 and later, Symbian UIQ 2.1 and preinstalled on all UIQ 3 phones, and a variety of Nokia, Samsung and Panasonic phones using S60. The newest release, Opera Mobile 9.5 is a beta version available to Windows mobile 5 and 6, and Symbian UIQ OSs as a free download.

Both 8.86 and 9.5 have support for JavaScript, AJAX, and CSS, and have features as a full fledge web browser. I.e. version 9.5 of the mobile browser use the same rendering engine as the desktop version 9.5, Opera Presto, hence have the same features as the desktop browser [73]. In addition, Opera Mobile 9.5 supports Widgets which can be used to developed web applications to function outside the browser window.

Opera have a mobile browser called opera mini which uses a server to compress web pages before they are sent to the browser. Opera mini does not support AJAX, but a “hybrid” of the Opera mini and Opera mobile is under development and is called Opera mobile 9.7. It can be referred to as a hybrid because it uses Opera Presto as the rendering engine and a server to compress the web pages by up to 80%, making the download time significantly shorter. The rendering engine should also have full score on Acid3 tests [74], making it at forefront of technologies [75].

2.3.7.5 Skyfire

Skyfire [76] is another cross device browser which is one of the fastest on the market today [77]. It is up to 10 times faster than Opera Mobile 9.5.1 Beta, and Safari. It uses a rendering server based on Gecko, similar to the Opera Mini, and also has support for Adobe Flash[78], Silverlight[79], and QuickTime[80] without any plug-in installation. At the moment Windows Mobile 5 and 6, and S60 devices are supported.

2.3.7.6 Mozilla Fennec

Fennec [81] is the replacer of the Minimo project in early 2008, which was Mozilla’s first attempt to produce a web browser for use on mobile phones. Fennec uses the same rendering engine as the desktop browser Firefox, Gecko, and hence has full support for JavaScript and AJAX technologies. Only Alpha release 2 is available at the moment, but the browser will probably be out on the market within reasonable time.

2.4 XMPP – the Extensible Messaging and Presence Protocol "aka" Jabber

This chapter describes relevant aspects of the XMPP standard, technology, usage and history. The facts in this chapter are based on author's understanding of relevant resources, most notably the RFC's referenced herein. Considerations and evaluations of the technology are left for discussion in later chapters of this report. The first part of the chapter will give the reader an overview of the technology, while the second part will explain the protocol specifics.

2.4.1 History

XMPP was first started as an open source project called Jabber [82] by Jeremie Miller in 1998. Miller was tired of dealing with the numerous proprietary IM clients in use on the Internet and started the Jabber project to standardize IM in one protocol. To be able to standardize IM; an extensible and versatile protocol was needed in order to cope with heterogeneous environments and software. According to the original ideas for the Jabber project [82] [83], the core components of an IM protocol should include:

- Open – fostering the development of third party client software
- Decentralized – anyone can run their own Jabber server
- Secure – Strong encryption, authentication and identity features to protect privacy, confidentiality and spam.
- Flexible – any custom data and payload is supported

Jeremie Miller launched the first Jabber server on January 4th 1999 and encouraged the open source community to contribute. The community responded by creating client and server software based on the Jabber protocol. [84]

The Jabber protocol was successfully drafted as an Internet standard in 2002 and the core specifications became RFC3920 [85] in the fall of 2004. The name Extensible Messaging and Presence Protocol was adapted to conform to Internet engineering Task Force (IETF) naming conventions, hence Jabber and XMPP is often used interchangeably. In addition to the core specifications, several extensions exist as Internet standards, amongst them, RFC 3921 [86]: XMPP Instant Messaging and Presence, RFC 3922 [87]: Mapping the XMPP to Common Presence and Instant Messaging and RFC 3923 [88]: End-to-End Signing and Object Encryption for the XMPP.

During the standardization process, the Jabber community founded an organization named JSF, the Jabber Software Foundation to coordinate the work of the growing community and as responsible for the development and standardization of the Jabber protocols. JSF sponsored the submission of the successful Internet draft of the Jabber protocol in 2002 and has continued its work on the Jabber protocol. In 2007 JSF changed its name and is now known as the XMPP Standards Foundation, or XSF.

Extensions to the XMPP protocol, called XEP's are maintained by XSF and can be found on their web site xmpp.org

2.4.2 Overview

XMPP consisting of the core and its extensions is a protocol made to enable streaming elements of structured information between two network endpoints in close to real time. XMPP provides a standard framework for exchanging XML data, and while it is mostly used for Instant Messaging (IM) clients and exchange of presence information conforming to RFC 2779 [89], it is capable of transporting arbitrary XML data as well. Figure 12 shows a multi server setup where connections might be established locally from client → server → client or between domains, from Client → XMPP server → XMPP server → client. Each server represents one domain.

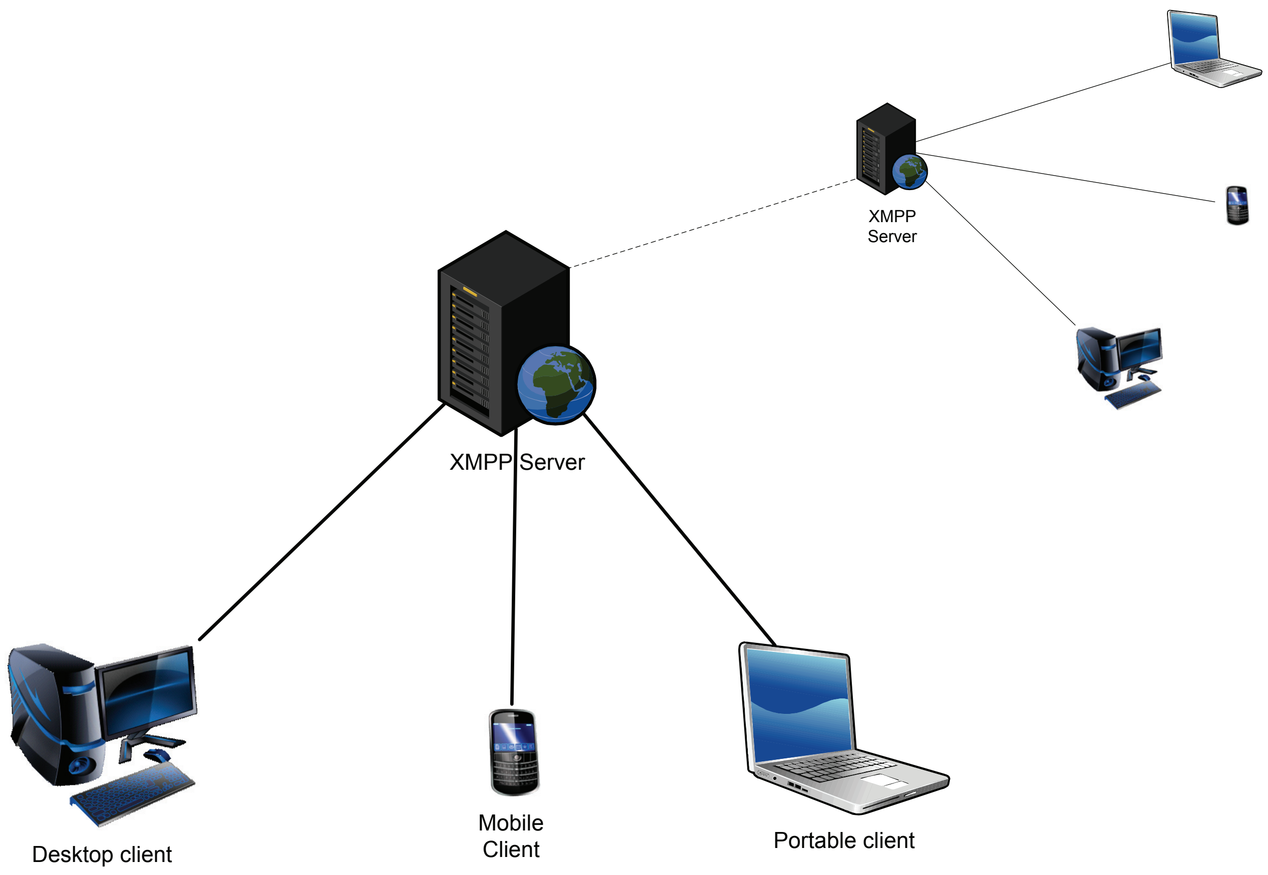


Figure 12 - Basic XMPP setup with multi server setup

The XMPP server maintains a database of all its users and their information. Information stored about each user includes:

- The name of the user
- The Jabber ID (JID) which is the unique identifier of each user has the shape of an email address: "username@domain" where the domain is the XMPP server for this user.
- The users current Presence information,e.g. status information like online/away/offline
- The time created
- The time of last logout
- Each user has a current list of friends, which in turn contain
    - The subscription state for each user
    - The groups to which the user belongs
    - The Last logout time
    - The nickname

The subscription state needs a little more explaining. This has to do with the way the XMPP protocol organizes the request and confirmation for new friends and is important to the understanding of the solution made in this thesis.

2.4.3 Subscription and roster management process

The following example will use the notion of a user as the local user and a contact about a remote user. Subscribing to a contact's presence information is closely linked to adding the contact to the roster of the user. There are four states of subscription [86]

- None – neither the user nor the contact receives any information about each other's presence
- To – The user has subscribed to the contact's presence information but the contact has not.
- From – The contact receives presence information from the user, but the user does not.
- Both – Both the user and the contact receives presence information about the other

Additionally there are several pending states in between these states. e.g. To + Pending In, which means the user is subscribed to the contact, and the contact has sent the user a subscription request, but the user has not answered yet.

The next few paragraphs will explain a "happy path" from subscription state "none" to subscription state "both" including the handling of roster states. The happy path assumes both parts of the connection accept the new subscriptions.

2.4.3.1 Adding the contact to the roster

The first step is to tell the server to add the contact to the roster. During this process three packets are transmitted.

- User sends: Set the contact in the roster
- Server sends: Set the subscription to none for contact
- Server sends: Contact was added to roster

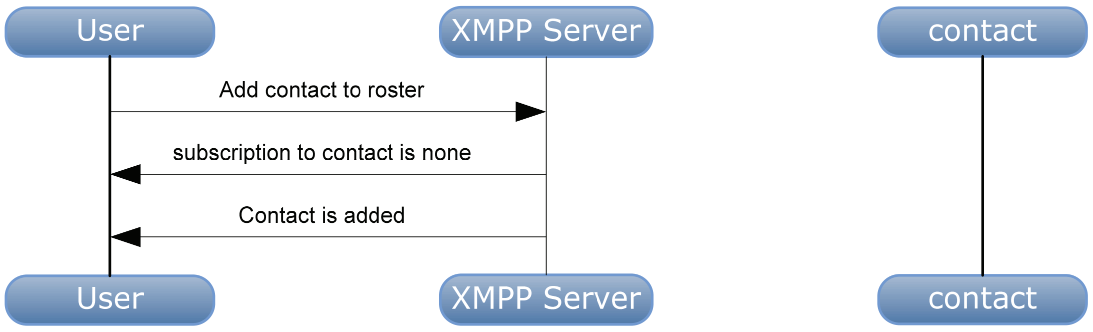


Figure 13 - User adds the contact to his roster

2.4.3.2 Subscribing to the presence information of the contact

The next step is to subscribe to the presence information of the contact, this involves

- User sends presence type subscribe to contact
- Server replies with a roster update: subscription is still none, with ask subscribe
- Server adds the users JID to the presence packet and sends it to the contact
- Contact responds by adding the user to its roster with a set message
- Contact sends a presence type subscribed to the user
- Server responds to the contact with the user set in the roster with subscription from
- Server responds to contact with: contact is added to roster

- Server forwards the subscribed packet to the user
- Server sends presence information. E.g. “Available” from the contact to the user
- Server sends a roster update, subscription is now to, to the user
- User sends a presence type subscribe, to confirm the reception of the subscribed (optional)

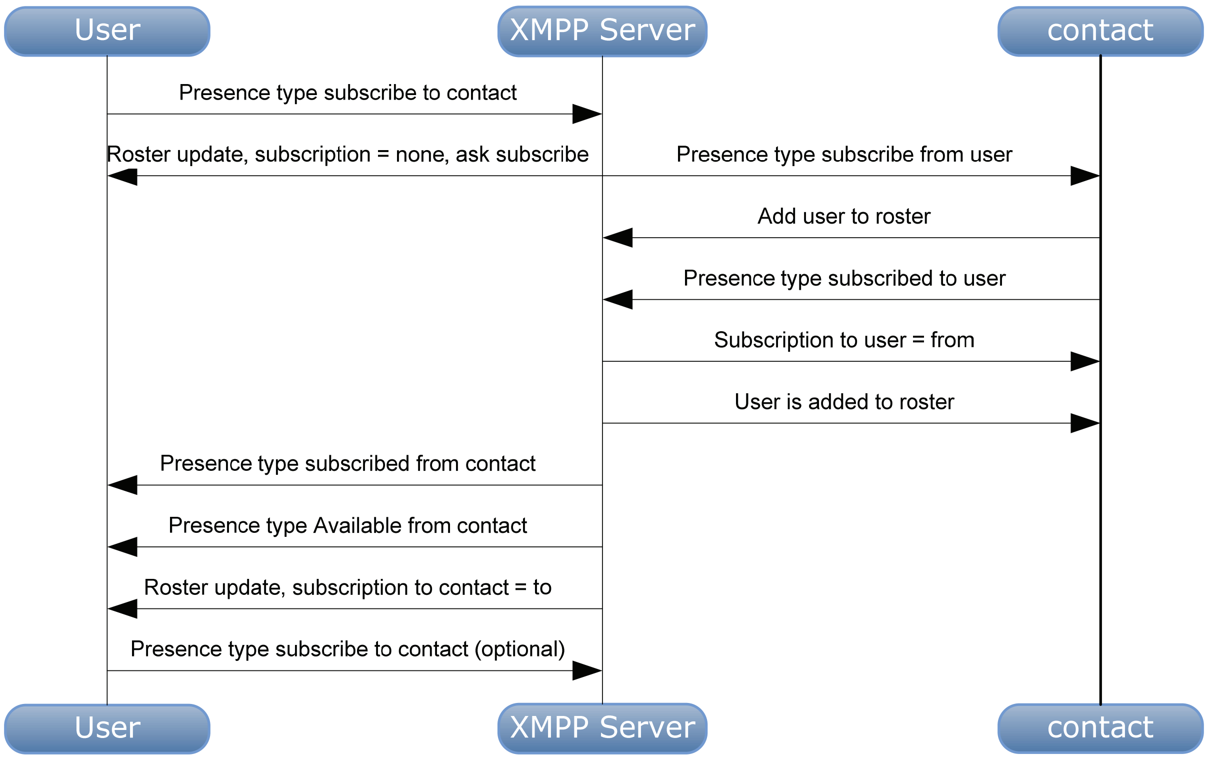

User
XMPP Server
contact
Presence type subscribe to contact
Roster update, subscription = none, ask subscribe
Presence type subscribe from user
Add user to roster
Presence type subscribed to user
Subscription to user = from
User is added to roster
Presence type subscribed from contact
Presence type Available from contact
Roster update, subscription to contact = to
Presence type subscribe to contact (optional)
User
XMPP Server
contact

Figure 14 - User subscribes to the presence information to the contact - the "happy path"

2.4.3.3 Obtaining a mutual subscription state

The following will continue building on the “happy path” above, e.g. both sides accept the procedure, and the user subscribes to the presence information of the contact. I.e. from the user’s point of view, the subscription state of the contact is now “to” and the user is receiving presence information from the contact. From the contacts point of view, the subscription type is now “from”. The contact gives away presence information, but does not receive any. The next step is to create a subscription type of “both” where the contact also receives presence information from the user. This procedure involves;

- Contact sends presence type subscribe to the user
- Server responds with a roster update, subscription is still from, with ask = subscribe
- Server forwards the subscribe message to the user, with the contact as a sender
- User replies with a presence type subscribed
- Server replies to user with a roster update, subscription is now both
- Server forwards the subscribed message to the contact, with the user as the sender
- Server sends a roster update to the contact, where subscription is set to both
- Server sends the current presence information of the user to the contact, i.e. presence available

- Contact should acknowledge the subscription state by sending presence type subscribe to server

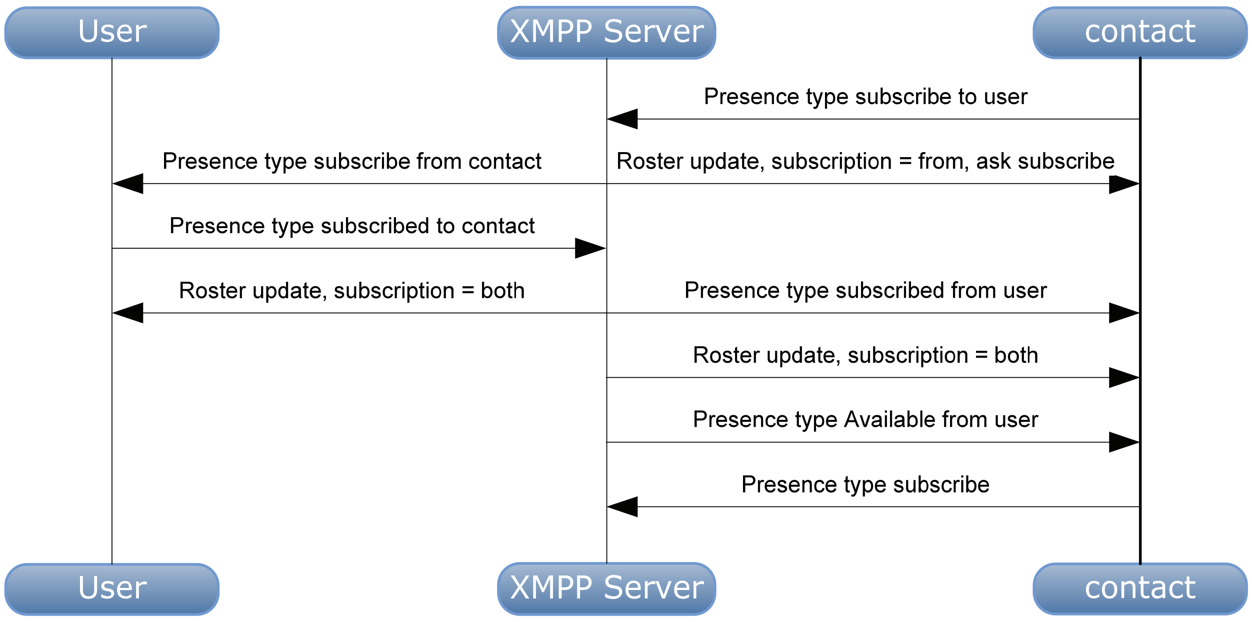

User
XMPP Server
contact
Presence type subscribe to user
Presence type subscribe from contact
Roster update, subscription = from, ask subscribe
Presence type subscribed to contact
Roster update, subscription = both
Presence type subscribed from user
Roster update, subscription = both
Presence type Available from user
Presence type subscribe
User
XMPP Server
contact

Figure 15 - Building on the happy path, to achieve mutual subscription state "both"

2.4.3.4 Cancelling subscription and removing contact from roster

There are many steps involved in completely removing a contact from the roster of both the user and the contact. There is however a shortcut to all these steps. The user might send a subscription type remove to the server, detailing the contact to remove.

- User sends a subscription type remove to the server
- Server sends a presence type unsubscribe to the contact from the user
- Server sends a subscription type to, to the contact
- Server sends a presence type unsubscribed to the contact from the user
- Server sends a subscription type none to the contact
- Server informs the user that the contact is removed from the roster of the user
- Server acknowledges that the remove is done
- Server sends a presence type unavailable to the contact from the user
- Contact sends a remove request to the server

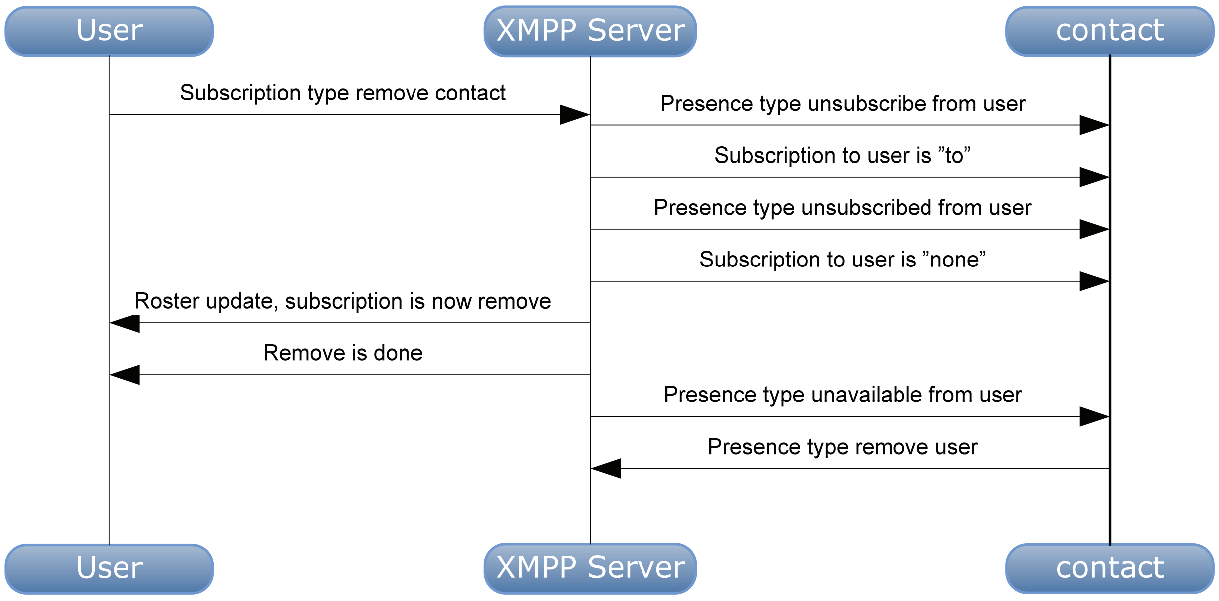

User
XMPP Server
contact
Subscription type remove contact
Presence type unsubscribe from user
Subscription to user is "to"
Presence type unsubscribed from user
Subscription to user is "none"
Roster update, subscription is now remove
Remove is done
Presence type unavailable from user
Presence type remove user
User
XMPP Server
contact

Figure 16 - Using the presence type remove to cancel subscriptions both ways

This has been an explanation of some of the most important aspects of subscription in the XMPP protocol, explaining how a simple scenario of subscribing to a contact and let the contact subscribe to the user. Furthermore it was shown how a mutual subscription could be ended. The next chapter will look at the XMPP core specifications.

2.4.4 XMPP Core Specifications

The core specifications of XMPP, as outlined in RFC3920 and RFC3921 comprise the following aspects, which will be explained in some detail in the next paragraphs.

- Jabber client
- Jabber server
- Presence and IM session Establishment
- Resource Binding
- Server Dial back
- SASL - Simple Authentication and Security Layer
- S/MIME Encryption
- Stanza Errors
- Stream Errors
- TLS - Transport Layer Security
- XML Streams

2.4.4.1 Jabber Client

A client is an entity which connects to a server via an XML stream. The standard port for this connection is 5222. It does so by authenticating itself with a set of credentials in a local account. For the duration of the client – server session, XMPP is used for the communication with the server and other clients connected to an XMPP server. A client can have multiple sessions with the server at the same time. Each session is identified by the use of the resource identifier portion of the XMPP address, e.g. <client@server/car> and <client@server/home>.

2.4.4.2 Jabber Server

A Jabber server’s primary objectives comprise; Managing XML streams with local clients and deliver XML stanzas to the same clients over the negotiated XML streams. In addition to this, the server is also responsible for the same XML negotiation with other XMPP servers. Unless otherwise customized by the administrator, XMPP servers communicate on port 5269.

2.4.4.3 Presence and IM Session Establishment

Presence and IM session Establishment is a vital part of the XMPP protocol and specifies the conformity to the basic IM and presence functionality defined in RFC 2779. The core of XMPP defines the transfer of XML streams, using TLS and SASL and the syntax of various message types. This is the foundation for creating many near-real-time applications. The Presence and IM session Establishment, while a part of the base specifications have been separated into its own Internet standard, RFC 3921. This enables the use of XMPP for generic near-real-time applications without necessarily implementing IM functionality. RFC 3921 describes the implementation of the Exchange of messages with other users, presence information, subscriptions to and from other users, managing items in a buddy list and blocking of communication from specific users. Further reading on the subject can be found in RFC 2779 which further specifies that a presence service and message service should be able to exist independently from the other, enabling the creation of applications implementing one, without necessarily the other.

2.4.4.4 Resource Binding

Resource binding describes the syntax for negotiating the resource part of the full Jabber ID or “JID” which is the trailing part of the identification of type <name@domain/resource>. This is a mandatory part of the protocol. By default the resource is set by the client in the initial negotiation of the XML stream, but it can also be set by the server, if the client wishes. The resource identifier enables a client to have multiple sessions with the same server at the same time.

2.4.4.5 Server Dial-back

Server dial-back is an authentication scheme for server to server authentication to avoid domain spoofing. It was developed as part of the early Jabber protocols and was included in XMPP primarily for compatibility reasons. It enables weak domain authentication of remote servers. For strong authentication SASL and TLS should be used.

2.4.4.6 SASL Simple Authentication and Security Layer

XMPP implements a profile of the Simple Authentication and Security Layer protocol. RFC 2222 SASL [90]is a generic scheme adding authentication to connection oriented protocols. XMPP implements a generic XML namespace that conforms to the SASL requirements.

2.4.4.7 S/MIME Encryption

S/MIME Encryption, specified in the Internet standard RFC 3923 defines a scheme enabling the XMPP client to sign, encrypt or sign-and-encrypt the contents of any XML stanza with S/MIME technology. Typically this scheme uses X.509 certificates. To conform to the IMPP workgroup’s common presence and Instant Messaging (CPIM) specifications, the format of the message must conform to CPIM Message Format. This ensures the interoperability with other protocols.

2.4.4.8 *Stanza Errors*

Stanza errors define errors which happen inside the XML stanza. This is the actual XML payload sent inside an XML stream, between client and server or between servers. Three types of stanzas are defined; <message/>, <presence/> and <iq/> with five common attributes. Stanza errors are similar to Stream errors, with one important difference; stanza errors are recoverable while stream errors are not. As a helper function when errors occur, the stanza errors contain hints to what can be done to alleviate the error. A receiver of a stanza might detect an error and will then send a stanza back to the sender with the same kind, but with the type set to the error value. The error stanza should contain the original XML, for the sender to validate and correct it before trying again.

2.4.4.9 *Stream Errors*

Stream level errors are sent by one of the participants in a connection, a compliant entity, most often a server. It is sent when the participant believes a stream error has occurred. Stream level errors are thought of as unrecoverable; hence the sender of the stream error must further send a closing stream-tag and terminate the TCP connection. A new connection would have to be created after a stream level error. This is also a security feature in the Server dial-back scheme. Conditions for stream errors are all defined in the Internet standard; RFC 3920

2.4.4.10 *TLS – Transport Layer Security*

In XMPP, TLS may be used for point to point security, securing the data stream between a client and a server or between servers. This necessitates the implementation of server certificates. The XMPP Standards Foundation has started an initiative to speed up the use of TLS in XMPP applications. They do this by offering free signed certificates to XMPP domain administrators for easy install in XMPP servers. If TLS is used, SASL authentication must follow. The SASL negotiation should use the certificate obtained during the successful TLS negotiation. It is important to discard any information obtained in an unsecure matter when TLS is successful. Because of the nature of TLS, compression of the XML stream is also supported.

2.4.4.11 *XML Streams*

There are two basic concepts involved in the sending and reception of structured information, XML Streams and XML Stanzas. The XML stream is a container for all elements of XML sent over the network. The XML stream is negotiated in the beginning of a session between two entities in the network. During this session, an arbitrary amount of XML Stanzas or other elements might be transferred over the stream. Examples of these elements are TLS and SASL negotiation or Stanzas of type <message/>, <presence/> or <iq/>. When the initiating side of a connection has negotiated an XML stream, the opposing side will respond by creating its own stream in the opposite direction.

2.4.5 XMPP - Relevant Extensions

The previous paragraphs described the base of the XMPP protocol, defined in RFC 3920 and 3921. The XMPP Standards Foundation also maintains a growing number of extensions to the base protocol. These extensions are not a mandatory part and may be implemented for special applications. This makes XMPP dynamic and up to date. The standardized base can be used together with extensions that enable using XMPP for applications outside the scope of the original protocol. The following paragraphs will look into some of the more relevant extensions for use in this thesis.

2.4.5.1 BOSH – Bidirectional-streams over Synchronous HTTP

BOSH [91], also known as HTTP Binding defines a transport protocol emulating a two-way stream between two network entities, e.g. a client and a server by using multiple synchronous HTTP request / response pairs without the need of polling or asynchronous chunking. This extension is in fact not a part of XMPP; the BOSH extension might be used to transfer arbitrary XML data over HTTP in an efficient manner. XMPP specific extensions to the BOSH protocol have been documented in XMPP over BOSH [92]. The usage scenarios of BOSH include users behind firewalls; where there are port restrictions prohibiting a regular TCP connection, or users of web based clients where all traffic goes through HTTP. The benefits of using BOSH over regular HTTP, which users a request response protocol are the responsiveness of the protocol combined with low bandwidth usage. The BOSH technique encourages the connection manager (server) to only respond to a request once it has actual data to send to the client. When the client receives the data it immediately responds. In this way the connection manager is most often holding a request which it can push to the client once it has actual data to send. By opening another connection, the client can send data to the server even when data is being pushed to the client. With this scenario, the latency is as low as a regular TCP connection, without the overhead of regular polling with HTTP, instead the overhead is only slightly higher than a normal TCP connection.

2.5 OSGI, Eclipse Equinox and Eclipse RCP - modular applications

One of the primary objectives of this project is to create a platform providing the ability to dynamically extend the services available to the user. In other words, the user should be able to update his application with new functionality without triggering a complete download and fresh install of the whole application. This has many advantages over a static application which would trigger many different packages available for download from the service provider. With a modular approach, a generic package is downloaded during the initial setup and additional packages are integrated later on. This is quite similar to what a Java 2 Enterprise Edition (J2EE) Application server provides, where plug-ins are added to a container at runtime and discovered automatically. The drawback of using J2EE is discussed thoroughly later on, but the bottom line is its complexity. This complexity is not necessarily an advantage on an end user system as it generates many new sources for user errors and pitfalls. This chapter is a glimpse into the world of an alternative solution the Eclipse RCP framework, utilizing standard Java (J2SE) frameworks. The purpose of this chapter is to enlighten the reader in the advantages of using a modular application framework like Eclipse RCP.

2.5.1 OSGi Frameworks

OSGi [93] is an open framework for creating applications with plug-in runtime functionality. It is not a tool in itself but rather a set of standards. The OSGi Alliance was established by Ericsson, IBM, Oracle and Sun Microsystems in 1999. The Alliance has a board of Directors and Officers with different roles in the organization. Technical work is organized in Expert Groups (EGs) while non-technical work is organized in Working Groups and Committees. The Alliance, through its Expert Groups has produced four major standards. The current standard is R4.1 released May 2007. Figure 17 show how the OSGi framework connects with different parts of a computer system. As shown, the OSGi framework is layered between the Java VM and the Bundles which comprise the application itself. These bundles are standalone, but may depend upon each other or other parts of the system to function properly

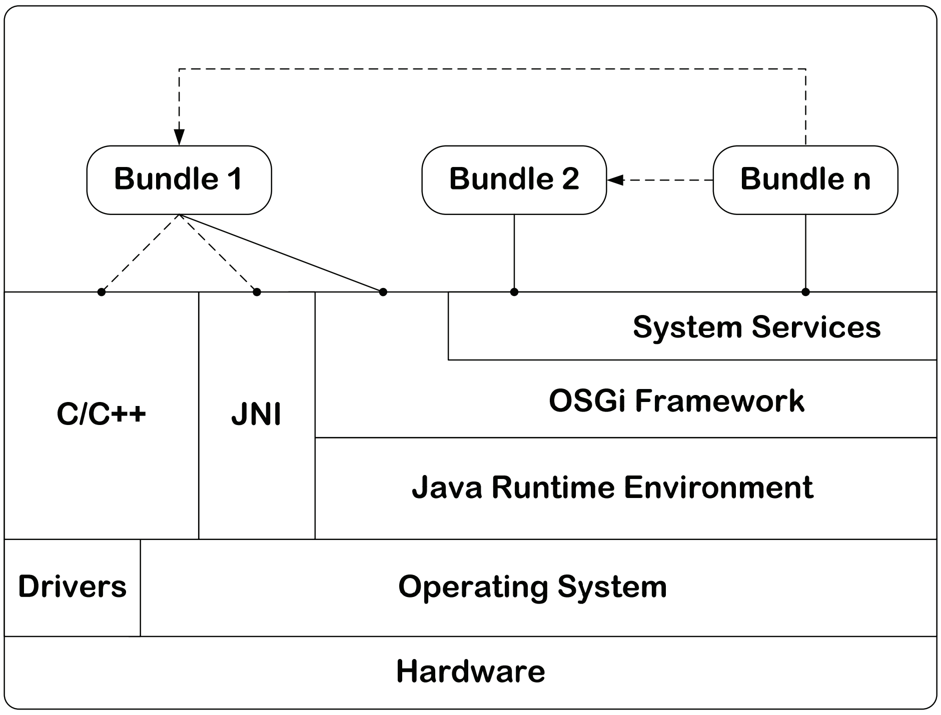


Figure 17 - OSGi framework in system layering

2.5.1.1 The OSGi layering

The OSGi architecture is divided into the following sub categories as shown in Figure 18 -

- Bundles – The bundles are the actual application made up of Plain Old Java Objects (POJOs)
- Services – This layer establishes a dynamic connection between the bundles
- Life-Cycle – The life cycle component takes care of installation, start, stop updates and uninstall of bundles
- Modules – Defines how a bundle can import and export code
- Security – all security is handled by this layer
- Execution Environment – Contains a definition of which methods and classes are available on a specific platform.

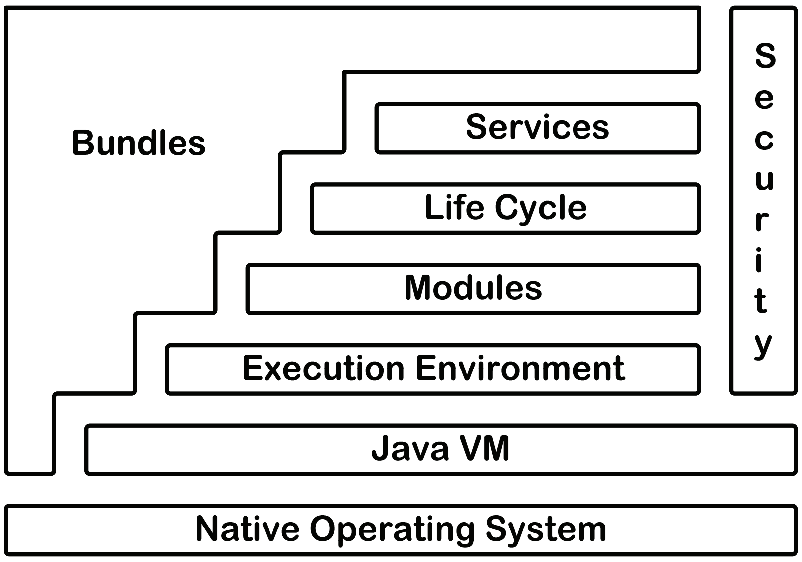


Figure 18 - Internal components of OSGi

Modules

The purpose of the module concept is to take a regular Java object and bundle it. This is a lot like creating a Jar file. The important difference is that, while in a jar file, everything is visible to the rest of the world; OSGi hides everything inside a bundle unless explicitly exported. A bundle which would like to use this exported entity would have to import it. This is one of the fundamentals of OSGi and the component which enables the modularity OSGi delivers. Because bundles are self contained and only collaborate in an explicitly stated manner, there are few sources of error and a high degree of modularity.

Services

Services are the OSGi solution to a familiar Java issue of writing a collaborative model with class sharing. For example getting instances of shared classes or generating objects of classes. OSGi utilizes a service registry to deal with this. A bundle creates an object and registers it in the service registry in one or more interfaces. Bundles' wishing to access this service goes to the service registry and gets a list of services available from specific interfaces or classes. It is also possible to listen for a specific service and get a call back when this service is available. The service registry holds a list of where the service is in use, which makes it easy to maintain an overview of in which bundles the service is used. How are services distinguished from one another? Services contain a set of properties to differentiate themselves from other services. This makes it possible to register more than one service under the same interface or class.

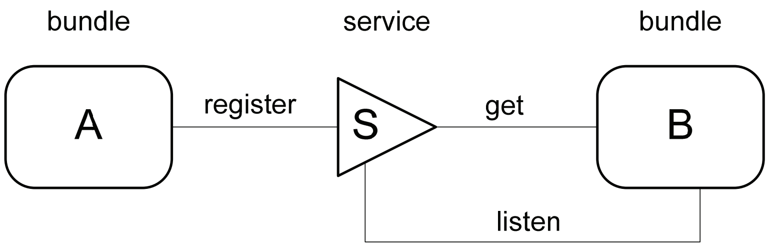


Figure 19 - OSGi bundle register

In real world scenarios, services come and go dynamically. OSGi has taken this into consideration and made its services dynamic. A bundle might withdraw its service at any given time. Bundles using this service must then drop their reference to the service. To help in this somewhat complex process OSGi uses frameworks like iPOJO [94], Spring and Declarative Services[current ref to osgi.org, more needed] to minimize the pain and maximize the gain. As it turns out, this dynamic approach is a good model of real world scenarios. As an example, consider a service responsible for maintaining a connection with a remote computer. The service will simply unregister if the computer is no longer reachable and register itself again when the computer is available again. This also means OSGi bundles do not require a preset start/stop ordering of their bundles.

Deploying OSGi bundles

Bundles are deployed in the OSGi framework. This bundle runtime environment has some similarities with a Java Application server, but the bundle runtime environment is collaborative in a way that an application server is not. Bundles run on the same Virtual Machine, and as explained can share code and collaborate with each other. Another advantage is the framework is standardized. Several APIs can be used to start stop install and uninstall bundles. As long as they conform to the OSGi specifications, they remain compatible to each other. Currently there are four open source implementations providing an API to the OSGi framework and several commercial ones.

2.5.2 Projects using the OSGi framework

2.5.2.1 Equinox

Equinox is an eclipse project based on the OSGi framework. Eclipse uses Equinox to provide the plug-in architecture on which Eclipse itself is built. [95; 95]

2.5.2.2 OSGi and Eclipse – Equinox

The eclipse Integrated Development Environment (IDE) uses plug-ins to extend its functionality to new areas. In fact, Eclipse itself consists of a large collection of plug-ins. Since version 3.0 the Eclipse project has based its plug-in architecture on the OSGi framework through the Eclipse Equinox project.

2.5.2.3 Eclipse RCP – Eclipse Rich Client Platform

Eclipse RCP is a plug-in framework based on the Eclipse platform. The RCP framework is a minimum set of plugins from the eclipse IDE to create a generic application container. I.e. RCP is Eclipse, but without the programming interfaces, IDE or Graphical User Interface (GUI). This makes RCP ideal for modular application development and extendibility. Eclipse RCP has a large user base and is well documented. [96; 96]

2.6 Information retrieval (IR)

Information retrieval (IR) [97] in this context is how unstructured material is found and how information, both content and metadata is retrieved and analyzed. In IR, the material is structured to meet the requirements of the user, and makes it possible to access different types of material using a query. Most users want to discover material of interest, while not using time to find the material. To provide the users with a simple and usable search engine, the technology behind the user GUI is explained in this chapter.

Accuracy and precision [98] are core concepts in IR. Accuracy is the percentage of the subject-material available which was actually returned.

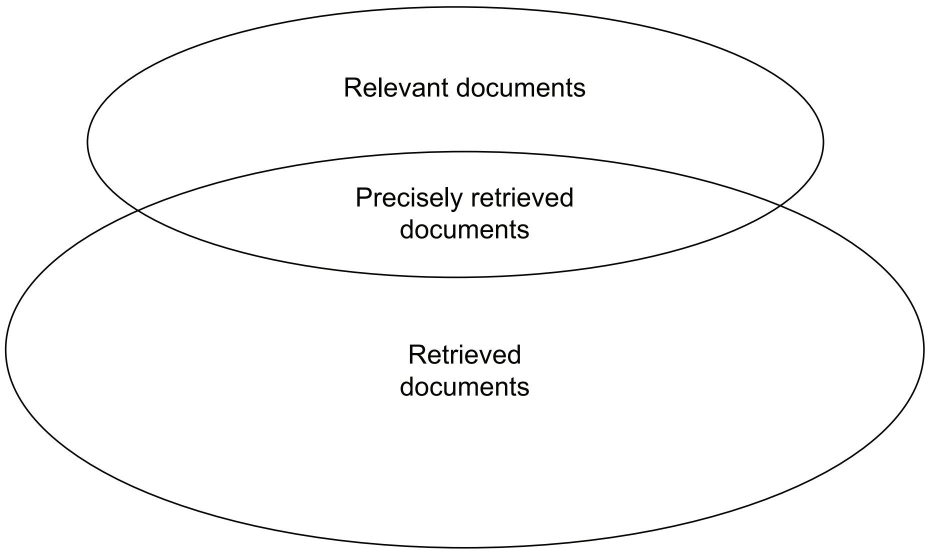


Figure 20 - Concept of precision

Precision is the percentage of the material displayed to the users which actually contain the queried subject. In a 100% precise search, the two circles in Figure 20 would cover each other.

When designing a search engine, you want to retrieve as many result with relevance as possible, but as few result as possible which has irrelevance.

2.6.1 Anatomy

A search application based on indexing is usually based on four entities:

- Index
- Query
- Search
- Display

Figure 21 show the four entities in a wider context, and it shows the possible anatomy of a search engine.

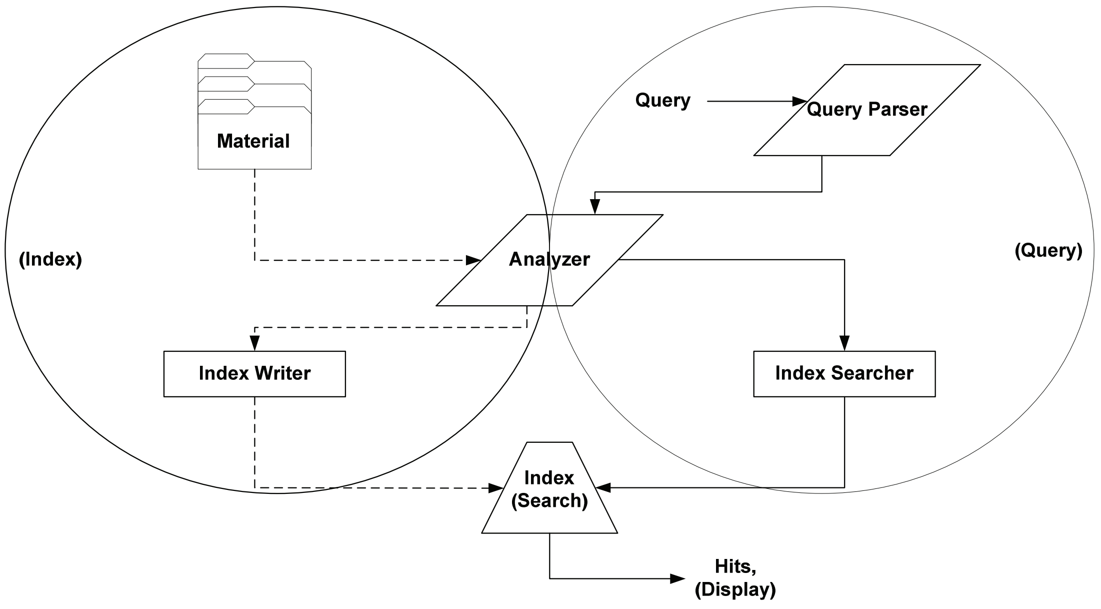


Figure 21 - Anatomy of a search engine

2.6.1.1 Indexing

Indexing is a process where content of materials is added to a system. The content to be indexed is retrieved from different material of an unstructured nature such as text documents, emails, etc, and broken into pieces which are stored in a database. The pieces, also called entries, are structured with a material identifier and a score defined by an analyzer. The score is basically a number describing the relevance of a given hit in the context of the material it was retrieved from.

There are two main types of indexing; forward index [99], also called a document index, and inverted index [100] [101]. In an inverted index the terms within a material takes center stage, while in a forwarding index the material itself take center stage. This means that in inverted index the terms that are to be indexed points to materials, while in forwarding index the material points to a list of terms contained.

The inverted index consists of several inverted lists, where the lists contain a term and docID pointing to materials containing the term. The list also holds information of how many times a term occurs in a material –called frequencies. The inverted index is also popularly named inverted file index [102].

In addition to the two fields mentioned, a position pointer can be stored in the list to point to the exact location of the term in the material (given by the docID). This inverted indexer is called full inverted index [103], and enables functionality for a "phrase search" in a material, because it includes the exact location of the term within the material. The list can also hold other contexts attributes such as font size etc. to make it easier to rank search results.

2.6.1.2 Query

On the opposite side of the index in a search engine, we have the query which is the user's input to the system. The query is parsed and analyzed before it is indexed using the same analyzer as the analyzer used to index the material.

2.6.1.3 Search

Searching is the process of linking the query and the database of the indexed content together. The indexed query and the indexed material are compared, and the matching result is defined as hits on background of the score, and displayed to the user.

2.6.1.4 Display

Display is not handled with the search engine or the IR mechanisms, but it is important to the user how the display is handled. The display, or the visual interface, is the feedback to the user based on the query. If the visual interface is difficult to grasp, the user experience is reduced. So it is important to have an easy to use and self explained display-service, making the user experience as good as possible.

2.6.2 Models

There are different types of IR models which structures and organizes material, making the retrieval as efficient as possible. There are three main model types; set theory-, algebraic-, and probabilistic models. In this report a set theory model called the Boolean model and an algebraic model called vector space model are most relevant models, and are explained in more detail in the next sections.

2.6.2.1 Boolean

The Boolean model [104] is in historical context the most used IR model. It uses a set of operators: AND, OR, and NOT to either present or absent the material. Due to Boolean logic no ranking of documents are possible, and often too few or too many documents are returned in response to the users query. The Boolean model can be represented with three circles, one for each operator. The composition of the operators either includes or excludes the material, see Figure 22.

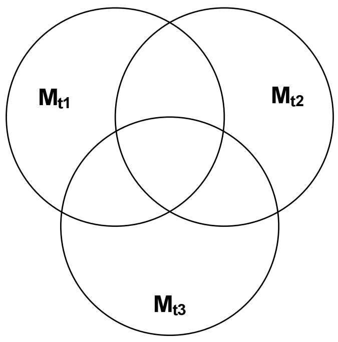


Figure 22 - The Boolean model

2.6.2.2 Vector Space

The Vector Space Model (VSM) [104] can make a hit although the query and the indexed material only have a partial match, and not as in the Boolean model. It can match degrees of similarity in order to make hits, and rank the hits based on the similarity. That is why it often is referred to as similarity models. VSM use vectors to indicate similarity. Basically the VSM uses a cosine function to determine the angle (described as Θ in Figure 23) between the document vector and the query vector. The cosine output is one if the two vectors are equal; this means that the closer the vectors are, the more relevance the document has. A simple illustration of the principle of the VSM is shown in Figure 23.

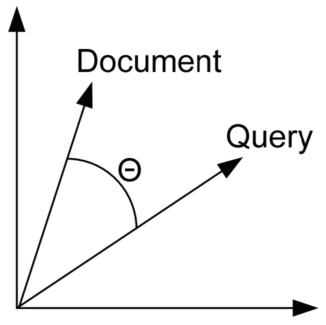


Figure 23 - Illustration of the VSM

2.6.3 IR Libraries

There are a few open source libraries with the functionality we need to build our search function. The ones with the most active community and best features are Lucene, Egothor, and Xapian. Lucene and Egothor are written in Java and have bindings to C++ and other programming languages. Xapian is written in C++ and have bindings to most of the programming languages besides Java, but Java should be supported soon.

We decided to use the Lucene API because it is the most widely used and best documented search engine API.

2.6.4 Lucene

Lucene [105] [106] is not a search engine application, but an open-source full-text search API using the Java framework. A description of the API is given in the chapter Libraries; in this chapter an overview of the functionality of Lucene is provided. In Lucene the indexing can be divided in four pieces; the index, documents, fields and terms. And in the following, this is the syntax used to describe the functionality of Lucene.

Lucene is a very modular framework, making it easy to add functionality and develop applications on top on this API. It uses a combination of VSM and Boolean logic to match a query to the indexed documents. The Boolean logic is used to filter out the documents of irrelevance before the VSM is used, making the system much more cost and time effective. VSM is used on the remaining material to find the most relevant documents using the logic of: The more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that document is to the query.

2.6.4.1 Scoring

The vectors in Figure 23 are based on a scoring schema which defines how a text in a material should be emphasized. The same scoring schema must be equally used in both the query and the indexer to give a comparable output. The scoring schema in Lucene is an advanced process of ranking documents, but the score is based on 6 factors [97] [106] [107] [108]:

- Term frequency (*tf*)
- Inverse document frequency (*idf*)
- Boost
- Normalization value of a field (norm)
- Coordination factor (coord)
- Normalization value of a query (queryNorm)

$$score_{q,d} = coord_{q,d} \cdot queryNorm_q \cdot \sum_{t,q} \left(tf_{t,d} \cdot idf_t{}^2 \cdot t.getBoost \cdot norm_{t,d} \right)$$

$$q = query, d = document, t = term$$

The *tf* is based on how many times a term is presented in a single document, and correlates to the term's frequency. The *tf* is computed between a query term *t* and a document *d*, ergo: $tf_{t,d}$. The problem with $tf_{t,d}$ is that all terms are considered equaly important when it comes to assessing relevancy on a query, or put another way, $tf_{t,d}$ treats all words equally in a document.

$$tf_{t,d} = frequncy^{\frac{1}{2}}$$

To differentiate or weight a word more than another, *idf* is used. *Idf* is a mechanism of attenuating the score if a term occurs to often in a document to give relevance. To compute the *idf* there are at least two methods: Using the collection frequency *cf* or the document frequency *df*. The *cf* takes all the documents matching the query to compute the score, while the *df* only compute the score based on the documents actually containing the queried term, df_t: . From the equation below, the *idf* has a low score if the term is frequent and a high score if the term is rare.

$$idf_t = 1 + \log \frac{N}{df_t + 1}$$

$$N = total\ number\ of\ documents\ in\ a\ collection$$

The boost factor is the users (or developers) way to influence the outcome of the score. The users can influence the query by boosting a term in the query, called *q.getBoost*. The user can boost the term by using the ^ character followed by a number to boost the term, i.e. "Lucene^3 is an API". The boost factor can also be used directly on indexed documents. There are two types of methods to boost documents; boosting the whole document, called *doc.getBoost*, or just a field in the indexed document, called *f.getBoost*.

The normalization value of a field is computed during indexing and stored in the index, and gives the number of terms within a field.

$$norm_{t,d} = doc.getBoost \cdot lengthNorm(field) \cdot \prod_{field\ \boldsymbol{f}, \boldsymbol{d}\ named\ as\ \boldsymbol{t}} f.getBoost$$

The coordination factor is based on the number of queried terms in a document. The score increases for each query term found in a document.

The normalization value of a query does not affect the ranking, but make the query score comparable.

$$queryNorm_q = \frac{1}{(q.getBoost^2 \cdot \sum_{t,q}(idf_t \cdot t.getBoost)^2)^{\frac{1}{2}}}$$

All the equations and parameters described above are the standard settings in Lucene, meaning that all parameters can be changed to fit a specific application.

2.6.4.2 Indexing

Lucene can only index textual formats [109] [110],which implies that external parsers must be used to allow Lucene to work with mime types different than plain text formats such as .txt and etc. A project called Apache Tika, a sub project of Lucene, offers an API making it easy to implement mime support in Lucene. This will be covered in the section of Apache Tika.

Lucene supports searching and indexing at the same time, but searches only for terms in the point of time the index was opened. Added or deleted documents to the index are not available until the next time a search opens the index. Lucene also support full inverted index, allowing a terms place in a document to be stored in the index.

As described earlier, the index in Lucene consists of documents, fields and terms. Figure 24 shows the conceptual architecture of how the indexing in Lucene is constructed. The name and value in Figure 24 is called a term.

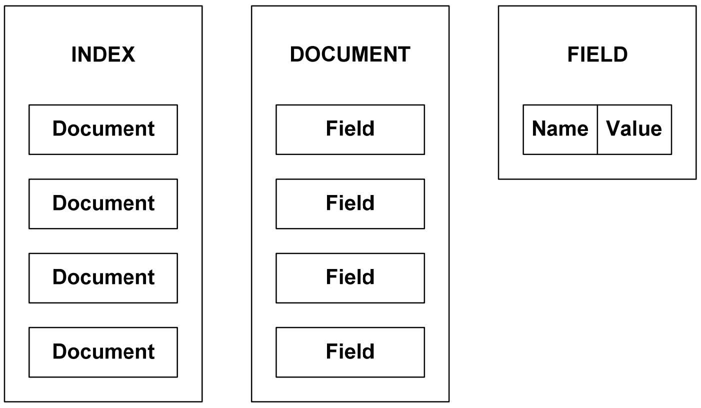


Figure 24 - Conceptual architecture of how the index in Lucene is constructed

2.6.4.3 Query

The query parser [111] analyzes the query and breaks it into pieces. There are two main types of queries: Single term and phrases and Lucene have basic support for these and additional methods of parsing the users query:

- Requirement, using +, –
- Wildcards, using *, ?
- Phrase, using “”
- Fuzzy, using ~
- Range, using TO (i.e. from a TO b)
- Boosting, using ^

- Boolean operators, using AND, OR
- Escaping special characters, such as: + - && || ! ( ) { } [ ] ^ " ~ * ? : \
  (To escape the characters, the \ must be placed before the character)

2.6.4.4 Analyzers

The analyzer must be used by both the indexer and query parser to make the outcome comparable. The analyzers convert field text into terms, which is used to match queries to material indexed. UTF-8 is the standard which Lucene use to store all characters, meaning all text to be analyzed must be retrieved using the correct encoding; otherwise Lucene will not be able to store them properly. I.e. in HTML a *meta* tag is used to define the character set, this tag must be handled and the data must be retrieved using the correct character set and then it is stored correct as Unicode Transformation Format (UTF)-8 in Lucene [112].

There are many possible features which can be built-in in the analyzer, such as white space and synonym analyzers, but the most relevant analyzers to this project are: Stemming, removal of stop words, and language specific analyzers. Stemming is the technology of retrieving the common root of a word and use this root word in both the query and index. The stop word analyzer does as the name indicates, it ignore all stop words, such as *the, an, a, it, is* etc. and leaves positional holes where the stop words are ignored. In Lucene both stemming and stop words implementations are integrated by using Dr. Martin Porters [113] analyzers, called: *PositionalPorterStopAnalyzer* and *PorterStemFilter*.

> "The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems" [114]

In the next section an open source project development of different kind of language specific analyzers are presented, also created by Dr. Porter.

2.6.4.5 Snowball

Snowball [115] [116] is a project of stemming words, which is an algorithm to process linguistic normalization in which the variant forms of a word are reduced to a common form. It supports a variety of languages, and a selection of the available languages is:

- English
- French, Spanish, Portuguese, Italian
- German, Dutch
- Russian
- Finnish
- Norwegian, Swedish, Danish

2.6.4.6 Apache Tika

Apache Tika is an open-source mime detection and extraction API which can be used together with i.e. Lucene. In Table 2 a overview of the supports mime types in Tika are listed [117]:

Table 2 - Mime types supported by Apache Tika

Text documents	.doc, .xls, .ppt, .rtf, .pdf, .html .xhtml, .txt, openDocument, and all Office 2007 documents
Audio	.mp3, .aiff, .au, .midi and .wav
Image	.bmp, .gif, .png, .jpeg and .tiff
Package	.tar, .jar, .zip, .bzip2, .gz and .tgz
Misc	.xml, .pst (Outlook mail) and .class (Java class files)

Apache Tika does not parse the materials itself, but detects the mime type and delegates the parsing to other existing parser APIs. Apache Tika can be thought of a wraparound of the other parsers, and have the functionality to detect the mime type and relay the material to the correct parser. How Tika function is illustrated in Figure 25.

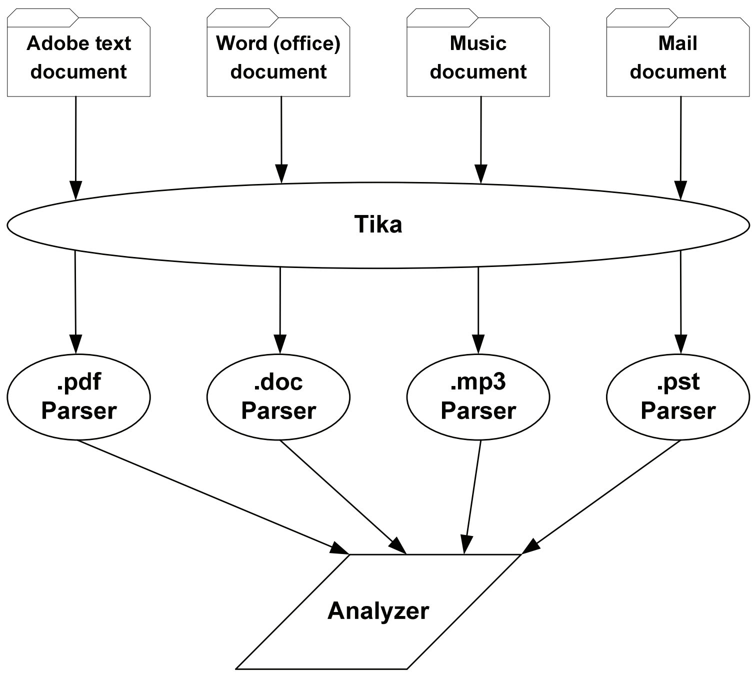


Figure 25 - Illustration of where the Tika project fit into the architecture.

Mobile Remote LAN Master Thesis

Part III – System Description

Design choices used to realize the service platform and the example service, the remote network search, are discussed in the design chapter. This chapter focuses on discussing the motivation of choosing the different technologies. For technical details about protocols etc., please read part two. Part three gives the motivation of why a technology or software is used, but does not explain the technology or software in detail.

The second chapter of part three, the Implementation chapter, explains how the prototype is implemented. Code snippets, screenshots, installation procedures, and configuration details are described in this chapter.

3 Part III – System description

3.1 Design

This chapter will discuss the different high-level choices taken during the creation of the service platform. There will be discussions as to which technology will be used and why. It will not go into details as to the implementation of the chosen technology, which can instead be found in the later Implementation chapter.

As described in the thesis introduction, the purpose of the service platform is to create a persistent connection across a diverse network topology and provide services to the user based on a modular architecture, regardless of location. The high level design of the system was described in the introduction and will now be taken one step further by looking at which solutions should be used for the various entities in the system.

3.1.1 System overview

The system overview describes the high level topology of the service platform. The system is designed to allow connections between user clients regardless of their physical location in the network. Three logical entities form the system.

- Home Servers (HS), are the hosts of services
- Remote Clients, are the consumers of services
- Network server (NS), relay the connection

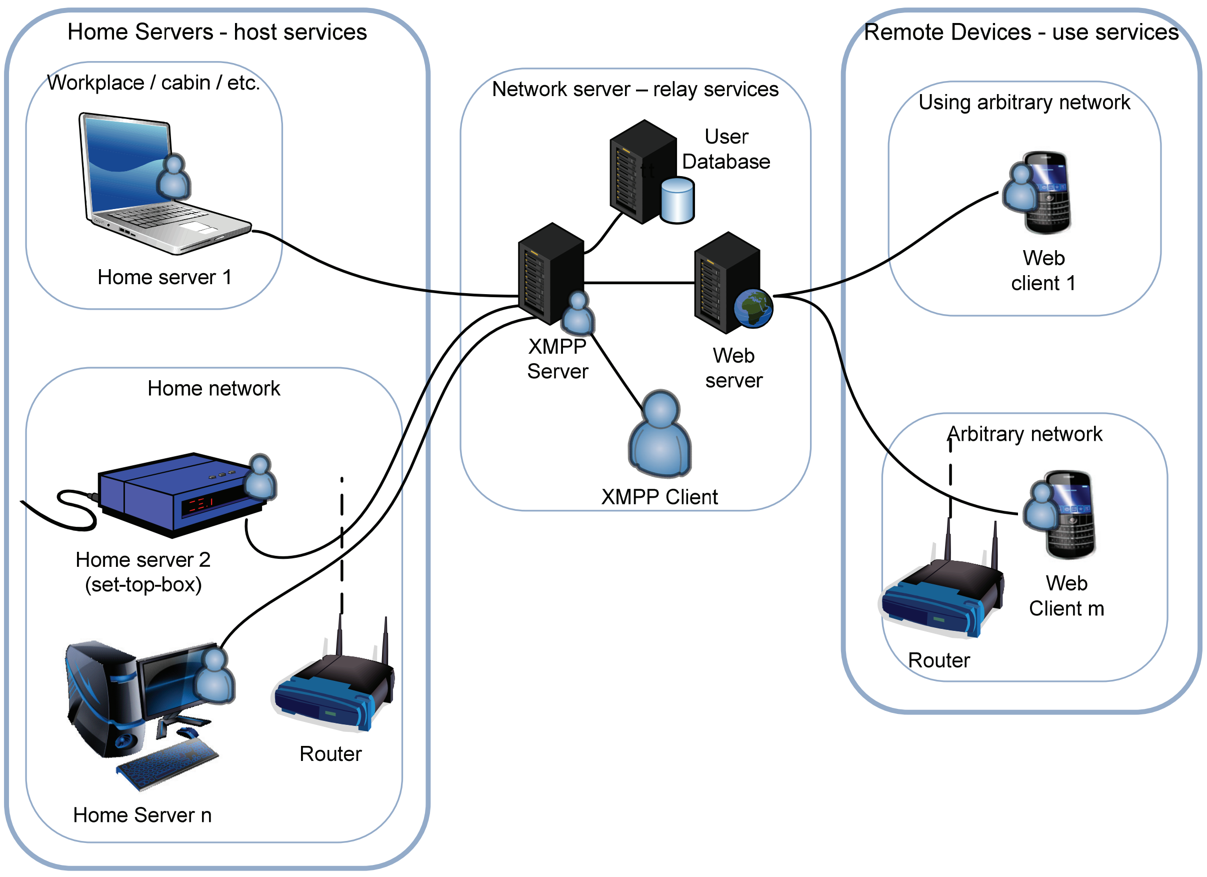


Figure 26 - Detailed system architecture

Figure 26 shows a setup where there are several different entities in the system at once. Two of the Home Servers reside in the home network behind a gateway, while the third Home Server resides in some other network. Two Web Clients are connected, one on a public IP, the other behind another gateway. The network server in the middle is shown with the database, web server, XMPP server and XMPP client. The figure is made to show how a Home Server might move around into another network and maintain the connection to the network server. This is possible because the XMPP server does not need to keep track of where the clients are, the clients are responsible for connecting to the server and relay that information. The routers are used to show how it is possible to connect to the network server from inside a private network without public IP addresses.

The choice of an IM protocol to solve the connectivity in this Master Thesis was inspired by discussions around how NAT and firewall traversal could be realized. It became apparent that a third party would have to be included to provide this connectivity. When this was realized, systems which already solve these kinds of problems were looked into. Amongst the studied applications were Google Talk, MSN Messenger, and Meebo, described earlier. These all used an intermediary server and a presence protocol to provide connectivity. The next realization was that most of them used a proprietary protocol to handle this. The exception was Google, which had sponsored the standardization work of a protocol called XMPP and implemented some of it in their own Instant messenger, the Google Talk.

The XMPP protocol is chosen to provide a communications framework and a service platform in this Master Thesis. The XMPP protocol is an open, standardized IM and presence protocol maintained by the XMPP Software Foundation. The protocol was chosen mainly because of its large user base, flexibility, and straightforward integration with many different software libraries and languages. Even though it is designed as an IM protocol, it can be used for transporting arbitrary data between participants.

The Home Server is the name used to describe an application installed on a computer or other kind of devices under the user's control. It provides persistent connectivity to the XMPP server in the context of acting as a host for services which the user would like to access from the remote client. There might be an arbitrary number of Home Servers in a user system.

The Web Client is used to describe a device where no application has to be installed. The Web Client uses a web browser to interface with the XMPP server. The purpose of the Web Client is to serve as an interface through which the user can access the services hosted by the home Servers in his system. There might be an arbitrary number of Web Clients.

The Network server is the term referring to the servers and applications that provide connectivity between the Home Server and the Web clients. It consists of four separate entities. The XMPP server is the core connection point where all XMPP clients log in, either directly or through a gateway. The web server is used as a gateway and user interface for the Web Client. The gateway works as a proxy and forwards traffic from the web interface to the XMPP server. The User database contains information about the users of the system, i.e. the XMPP clients; both remote and desktop users are stored here. The XMPP client residing on the network server is used to access resources on the network server itself. Resources might include file storage and/or file transfer. The following chapter explains the different entities in this system and how they were designed to achieve the functionality described here.

3.1.2 Network Server, NS

To achieve remote connectivity to a home environment, a server with a fixed address is used to provide address discovery and solve firewall and NAT implications. The NS is the connection point between HSs and remote devices.

The NS is a collection term for all the servers and applications working together to provide connectivity between the HSs and remote devices. The NS consists of the following parts:

- Web server
- XMPP server
- XMPP client
- Database

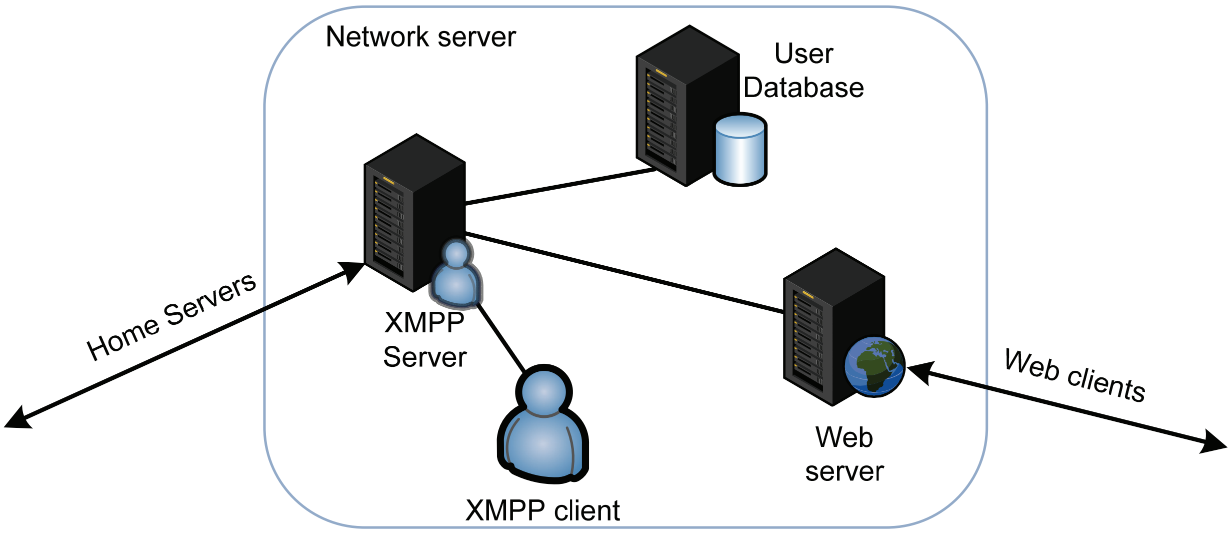


Figure 27 - High level architecture of Network Server, NS

An XMPP session with both the HS and the remote device must be set up before connectivity is achieved. Both the HS and the remote device must setup its own session to the NS on its own initiative, before the XMPP server can achieve connectivity between the entities. The XMPP protocol handles the initial session setup and the connection between the two devices.

Because of the same origin policy in browser side programming languages, a relationship between the XMPP server and the web server must be achieved. The web page is downloaded from the web server which contains the XMPP web client. If the XMPP web client tries to connect directly to the XMPP server, an error due to the same origin policy will occur, because the XMPP web client is downloaded from the web server and tries to access the XMPP server. To allow the connection, all XMPP communication between the web client and the XMPP server must be relayed by the web server in order to preserve the same origin principle.

3.1.2.1 Web Server

The web server is used to make a connection between the remote device and the XMPP server. The web server holds a web portal which allows the remote device to download and log in to the XMPP server. The choice of using a web portal, instead of an application software installed on the user‘s personal device, is made due to compatibility and usability considerations.

Personal devices are numerous, and have no common standard of how an application should be developed. It is possible to create a native application for each of the different mobile OS’s. However, this is a daunting task, because of the number of OS’s and the number of mobile devices. In addition, using a browser interface maintains the same user interface across different devices and is more user-friendly because no application needs to be installed. This solution was made and the web portal is used to interact with the rest of the service platform. To host the web portal, Apache2 [118] is used as web server because it is available in the Ubuntu 8.10 server which is used as the OS on the NS. Additionally it is the most commonly used web server on Linux systems [119].

The solution of using a web portal eliminates the need for OS compatibility. However, different browsers can still act differently to HTML and JavaScript. Even different versions can behave differently on the same content. Another issue in the browser compatibility, which occurred in this project, was different handling of XML documents by different browsers because they use different XML parsers. To use the parsers, different commands must be used to achieve the same functions e.g. in IE and Firefox. This means that the web page must have some functions to recognize the different type of browsers, and direct the browser to the correct command chain. Another example is the XMLHttpRequest object, this object is initialized differently by i.e. Firefox and IE.

Another deviation in the browser compatibility, which occurred in this project, was different outputs of an XML document. As mentioned earlier, XML documents can be different depending on the browser used. The XML document itself was the same, but the header, or the first tag indicating the XML document was different when it was received by the HS.

These examples show that despite the standards made by W3C, considerations regarding the user-agent/browser type must be made to achieve cross platform connectivity to the home environment.

Capabilities

The web portal must be in connection with the HS, and due to the protocol choice of XMPP, it must support the XMPP protocol for presence and message exchange. The XMPP protocol, together with AJAX, makes the web portal a dynamic placeholder of the user’s services, and provides the user connectivity to the home environment in an easy and simple way.

It is important to have a simple web portal, allowing the end user to use the portal with ease. The welcome screen should provide a basic login form, and a download site for the HS application. If the end user chooses to log in, all HSs, both available and unavailable, should be listed. If the user connects to a HS, all available services to that specific HS must be shown. The user must be able to select one of the services, and use the service from within the web portal. The user must also be able to log off the XMPP server from the web portal.

Browser requirements

Due to differences between mobile browsers and desktop browsers, even though the gap is decreasing, the web portal should use web technologies supported by most modern mobile browsers. I.e. Flash is widely used by web applications, and most desktop browsers have flash compatibility. However, most mobile browsers do not support Flash; therefore it should not be used in mobile browser applications. JavaScript on the other hand is supported by most new mobile browsers.

The portal is built up by using JavaScript, and the browser must support JavaScript to work. If JavaScript is disabled or not supported, the functions in the portal will be unavailable. The browsers must also support XML parsing and XMLHttpRequest. The portal supports both the ActiveX object used by IE, and the DOM parser/ XMLHttpRequest object used by Firefox and most other browsers.

The portal is not restricted to mobile devices, and can be accessed by any browsers meeting the following requirements: A user-agent with JavaScript activated, support for at least one of the two XML parsers described in the last section, and the ActiveX or XMLHttpRequest object.

3.1.2.2 XMPP server and client

To use the XMPP protocol, an XMPP server must be used to achieve connectivity; it is used to connect XMPP clients using the XMPP protocol. When an XMPP client logs in to the XMPP server, the presence to that specific client is changed, and a message is transmitted from the XMPP server to all clients in the clients' roster. To make a connection between the HS and the remote device, both parties must thus support the XMPP protocol.

There are many different types of XMPP servers available, both proprietary solutions and open source projects. The proprietary solutions are not considered in this report due to the cost, but three of the open source projects are reviewed.

To access the XMPP server using a web interface, which needs a persistent connection to offer connectivity between the entities, a method of imitating long-lived TCP connection was desired. The BOSH protocol [91] uses a sequence of requests and responses to make a reliable connection, and thus hides the potential short-lived Internet connection. The BOSH session holds the connection until the server times-out or ends the session. The cost of a more reliable connection is higher latency and overhead of HTTP requests.

When using a BOSH session to connect the remote device to the XMPP server, a BOSH connection manager is necessary. There are two types of connection managers: Stand-alone and built-in. The stand-alone connection manager is more flexible and support connections to external XMPP servers. The built-in connection manager makes the development faster and easier, and it should be more efficient and compatible due to development by the same vendor [120]. To focus on the main challenges, it was decided to use a XMPP server with a built in BOSH connection manager to use as little time as possible to get the backbone system up and running. The following servers fulfill the requirements of an XMPP server:

- Tigase [121]
- Ejabberd [122]
- Openfire [123]
- A more complete list can be found at http://xmpp.org/software/servers.shtml

These three XMPP servers could all have been used in this project, but Openfire was selected because it has the largest and most active user group. Openfire is easy to setup and has a modular platform which allows deployment of plug-ins available from a database. Alternatively, plug-ins can easily be created due to the open source development community to fit the project needs. It is available in Linux, Windows, and Mac OS X, has a connection manager built-in, and supports the use of an external database. Openfire is built up by extensions to suit the different needs, but the core of the application is illustrated in Figure 28.

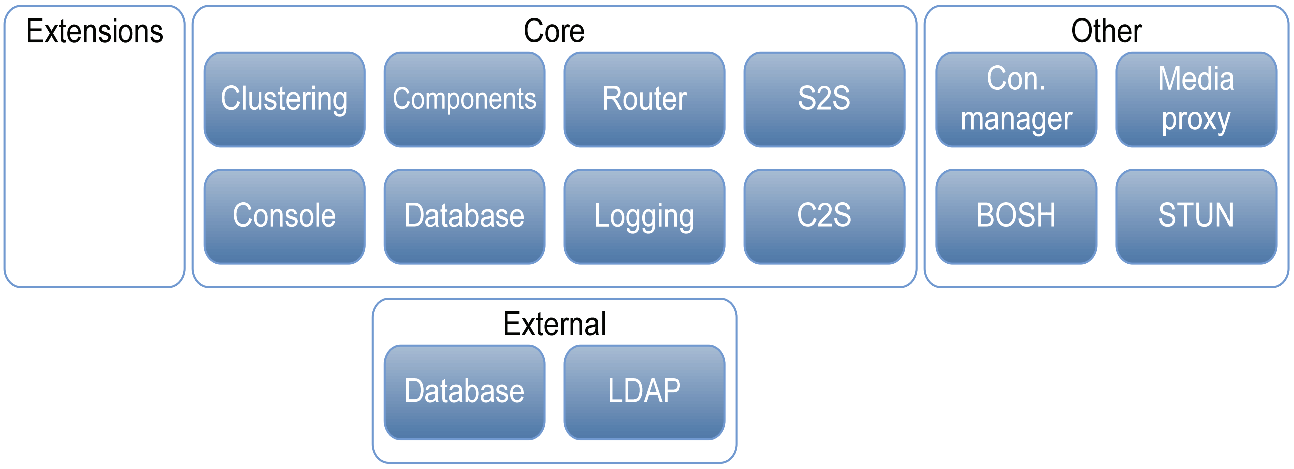


Figure 28 - Openfire architecture

In addition to the XMPP server, an XMPP client is deployed on the NS. The XMPP client is used to access local resources on the NS over an XMPP session. The XMPP client is specially designed with the search service in mind, but it is not restricted to the search service. It can be an important implementation in future services.

3.1.2.3 Database

Openfire supports an external database as illustrated in Figure 28, and it uses an open database schema to store information such as credentials, session, roster, group, and other type of data. The schema can be exported by Openfire to the following external databases [124].

- MySQL, GNUv2
- PostgreSQL, BSD
- HSQLDB, BSD
- Oracle, proprietary
- Microsoft SQLServer, proprietary
- IBM DB2, proprietary

Only MySQL, PostgreSQL, and HSQLDB are open solutions and therefore the only servers available to be used in this project. Due to knowledge of MySQL servers from earlier projects, MySQL was selected to work as the external database in the NS.

3.1.3 Remote device

The remote device can be a mobile phone, PDA, a desktop computer, or any other device with a web browser supporting Ajax technologies. When logged in, the Remote Device becomes the host of the Web Client. To access the system, a web page must be downloaded from the NS and a valid username and password must be used to log in. The web page is rendered and parsed using the web browser available in the device. There are no installations or configuration procedures, the only user interaction should be to start the web browser, type in the address to the web portal, and log in.

3.1.4 Home server, HS

The HS is a core part of the setup in this thesis. The purpose of the HS is to host services and enable seamless communication across network boundaries. It should be built as a modular platform to allow arbitrary services to be added as plug-ins, shown in Figure 29. The HS should be installed in a computer or other device in the user's possession, acting as a host for new services. The services might be located in the device or connected to the device and the Home Server over a proprietary protocol.

3.1.4.1 Choosing a platform

The idea is to create a platform which can be installed in the user's home computer or other devices. The platform should be able to provide connectivity regardless of user location. Additionally, the platform should be dynamically pluggable, i.e. it should be able to extend itself with new functionality. This new functionality should be able to use the connectivity provided by the HS to give the remote user some service. The main features needed for the HS are:

- Modular Architecture able to extend platform with new functionality
- Platform should be easy to distribute and install by end users

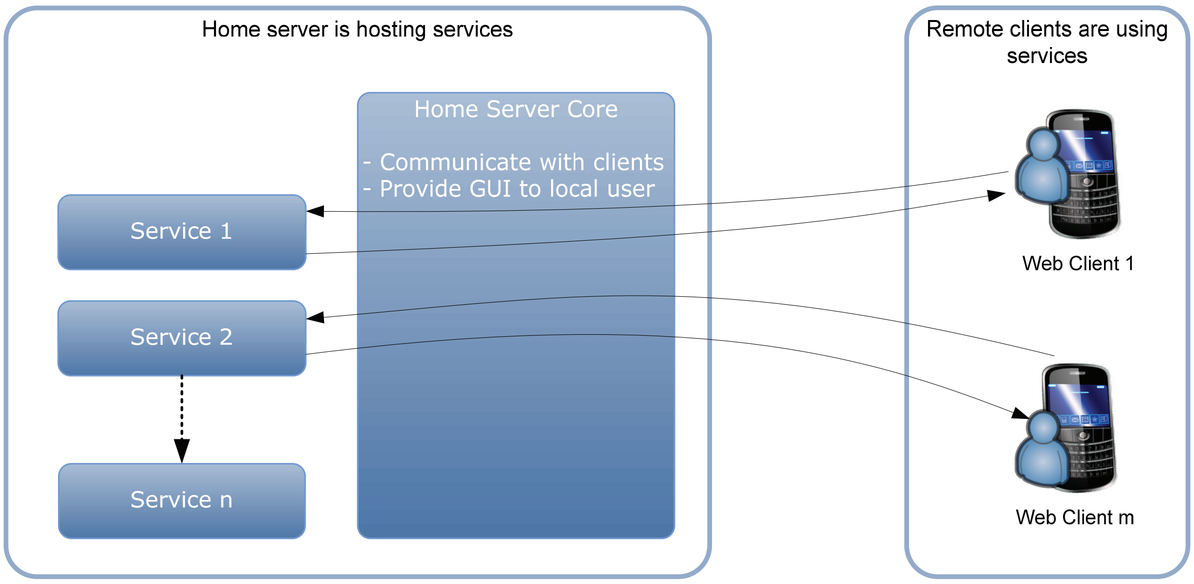

Home server is hosting services
Home Server Core
- Communicate with clients
- Provide GUI to local user
Service 1
Service 2
Service n
Remote clients are using services
Web Client 1
Web Client m

Figure 29 - Home Server logic structure

These features were considered when deciding how to create the HS application. The most important part of the design of the HS was to find a modular application framework. Early in the decision process it was decided to use the OSGi Dynamic module system as the framework. Several implementations of the OSGi framework exist. The Knopflerfish [125], Eclipse Equinox [95] and Apache Felix [126] projects were all considered.

In the end Eclipse Equinox became the framework of choice. There are several reasons why the Eclipse Equinox OSGi implementation was chosen. The framework is very popular because it is closely connected to the Eclipse IDE. In addition to this, the Eclipse project has a separate branch project which uses Eclipse Equinox to enable generic application development. This project, called Eclipse RCP, is the tool used for all the Java development in the thesis. It has a steep learning curve, but is quite well documented and has a large user base. Eclipse RCP enabled the creation of a dynamic, modular application as shown in Figure 29.

With a plan for the modular framework in place, some of the next parameters were given. Eclipse and OSGi rely on Java; therefore Java was used as the programming language. Several XMPP libraries were available in Java. The chosen library, "Smack" from Jive Software was considered the most mature with the right features to match the use cases mentioned in the introductory chapters, in addition to good quality documentation. Jive Software is the sponsor of the IgniteRealtime community and maintains Smack as an open source community project. The community also hosts the Openfire XMPP server used in this thesis.

3.1.4.2 Architecture and Message Exchange

Eclipse RCP was chosen to host the Home server. As described in Figure 29, the remote clients are expected to request a service from the Home Server; the Home Server should react to this message and respond to the client in an appropriate way. Since the home server is modular, it must include a relay to forward messages to the loaded services. The services should process the message and respond via the Home Server to the client. The protocol for enabling this is described later in this chapter. The scenario is shown in Figure 30.

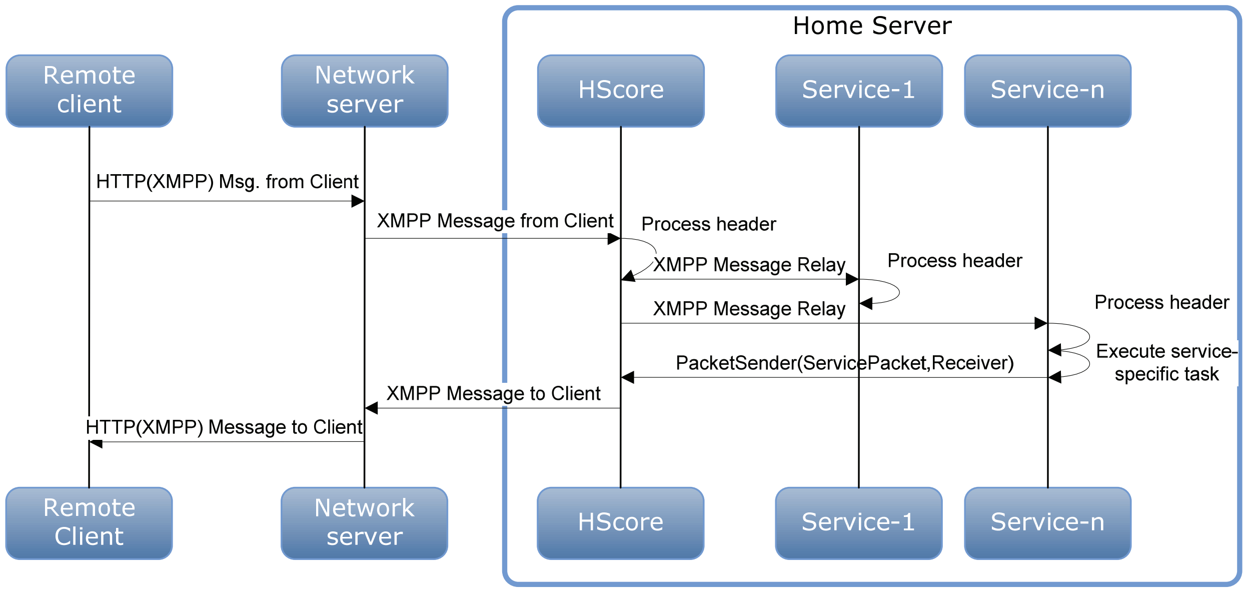


Figure 30 - High level message exchange

3.1.4.3 *Home Server modularity*

The use of the Eclipse RCP enables the creation of services as Eclipse Plug-ins. These plug-ins' can be dynamically loaded into the core Home server, called Hscore.

The Eclipse RCP platform relies on extensions and extension points to extend its capabilities into new areas. A plug-in defines an extension point in its plugin.xml file and registers it with the underlying OSGi framework. Similarly, plug-ins which would like to receive notification whenever the extension point is called, registers an extension for this extension point. Every time the extension point is called, it relays the message to any plug-ins extending this extension point. This feature is used extensively in the home application.

The core of the application will receive XMPP messages from the network containing an XML packet with a specific starting tag. If the tag is not targeted at the core itself, it is sent to the no.telenor.hscore.PacketHandler extension point. Any plug-in registered to extend the interface IPacketHandler, will receive the packet as shown in Figure 30. A filter in the plug-in then decides whether or not it should interpret or discard the packet. This isolates the core of the application from all the plug-ins. Because the core application does not have a static knowledge about the different services, it is very easy to extend the application. It makes it possible to develop a new plug-in without understanding the internals of the core application.

3.1.4.4 *The plug-ins*

Eclipse RCP applications are organized in plug-ins, essentially normal Java jar files, which are controlled by a plugin.xml and a manifest.MF configuration. These configuration files control the lifecycle and dependencies of the plug-in. In other words, the configuration files controls which plug-ins must be available for the successful execution of the specific plug-in. For the purpose of this application, a naming schema is developed to visualize the different parts of the system and to separate the core application from the add-on services. The third party libraries in use are named as sub packages of their respective plug-ins to identify which plug-in they belong to.

The core application: no.telenor.hscore

The hscore plug-in is the core part of the application, acting as an enabler between services and users by implementing the XMPP IM protocol.

Hscore provides an extension point for services to connect to. When a service extends the extension point, the users can communicate with the service through hscore. The functionality of the service plug-in is programmed in regular java which is bundled inside an Eclipse plug-in.

The XMPP API: no.telenor.hscore.org.jivesoftware.smack

The smack plug-in consists of the open source Smack XMPP library from jivesoftware.org, packaged as an eclipse plug-in. This will be used extensively to provide the XMPP communication in the no.telenor.hscore plug-in and does not contain any code except the libraries themselves.

3.1.5 Protocol

The communication between different entities in this system needs a defined syntax, a protocol to enable accurate and concise communications. At the same time, the choice of protocol should not limit the application to use fixed tags or the possibility to extend the protocol. Because the system created during this thesis should be able to extend itself into new areas and new applications, it needs a flexible protocol. The XMPP protocol, on which the communications channel is built, is using

XML. XML meets the requirements of being an extensible protocol, because the tags and meaning of the tags are set by the developer.

The data created by this application is transported inside the defined XMPP Stanza type <*message/*>. For all practical purposes this implies the protocol which is described in this chapter, is actually based on a chat session between entities in the XMPP network. This makes it very easy to debug, since any standard XMPP based chat client might be used to send and receive messages to the system, as long as it is authenticated.

In the design of a modular system, it was decided to use an extendible protocol which takes into account further expansions of the system; therefore the protocol is very simple. Basically, it consists of two parts, a service descriptor and its payload. The first tag, called the service descriptor is read by the receiver in the core application. If the tag matches that of the core, it is processed further; otherwise it is forwarded to the services. When the service descriptor tag does not match the one for the core, the payload of the packet is not read by the core application. Hence the services might develop their own protocol inside the payload of the packet without altering the core application itself.

The forwarding of messages is taken care of by the OSGi implementation of the home application by simply sending it to all registered plug-ins of a certain kind. The services are responsible for filtering out the packets of interest. This approach is used because it enables the core application to be unaware of the services it hosts. This approach should be communicated to the plug-in developers, since it implies all plug-ins essentially might read each other's packets. This could be circumvented by securing the internal payload of the packet in some way. The purpose is to leave this up to the plug-in developers, since the core application is not aware of the specifics of the loaded plug-ins.

3.1.5.1 Protocol description

The following is an outline of the protocol

```
<?xml version="1.0"?>
<body>
            <MessageType>BogusService</MessageType>
            …..Service specific payload……
</body>
```

As shown here, the MessageType is "BogusType". The core application parse this tag, finds out it is not of type "Communication" and therefore forwards it to the registered services. The services will then have to filter out the <MessageType> they are interested in. If one of the services uses the "BogusType" message, the service can process the message randomly. The payload of the packet, e.g. the part after the ending </MessageType> tag and before the </body> tag can be constructed whichever way the service finds best. For the example service made in this thesis, a similar XML protocol has been set up and used. This is described in the implementation chapter.

3.1.6 System bootstrap

System bootstrap explains methods of adding users to the service platform. The ability to seamlessly add new users in an orderly fashion might be necessary to get users to the system.

All the three A's: Authentication, Authorization, and Accounting are important when creating a seamless and user friendly solution. It would ease the planning of a system bootstrap method if the business model for the system had already been decided. In all scenarios it is important to create a solution which can be adapted to several business models.

The following will be a short description of the scenario under which the system bootstrap will be carried out. The system is based around the use of an IM presence server. This server has the ability to store some information about its users, most notably; their credentials and group affiliations. How users are added to the system must be decided by the administrator. By default the IM server, in this case an XMPP server allows everyone to add themselves as users and lets them form groups with each other. Fortunately, several configuration choices exist. It is quite possible to separate the user storage into a standalone system, e.g. a MySQL database. By doing this, it is possible to manage the database separately from the IM server and hence have full control over the user base.

3.1.6.1 Users administer themselves

In scenario where no central AAA authority is established, all Authentication and Authorization is performed by the IM server. The server makes sure the users are who they claim to be, but does not link this up to a real world person. This design would foster a somewhat more open solution, where the provider does not have full control over the users in the system, much like other public IM services. Accounting is not so easily managed in this scenario, since anyone can create users. If accounting is needed, some additional information would have to be requested when registering users, e.g. credit card information or address. The difference between this service and a typical IM network is that instead of focusing on contact with their friends, the system would focus on being friends with the devices of the particular user. In this scenario, anyone might create their own client software to connect and authenticate to the system.

3.1.6.2 Use a separate interface for user administration

Because the IM server relies on a separate database for storage and retrieval of the user information it is possible to create an AAA solution without involving the IM server in the administration of users. The IM server could be restricted from adding, removing or updating users and would simply care about the users currently in the database. In this case the owner of the system could create his own proprietary solution for user administration. A simple solution would be to have a member's site where users would administer their subscription, add new functionality and new devices to their list of friends. The site would connect to the database and update the information about the user without the knowledge of the IM server. A billing solution might connect to the same database and charge the user on behalf of the number of subscriptions, the number of services or some other scheme. The IM server would discover the database update the next time it accessed the user information. Creating this solution will take some more time and effort than the open suggestion mentioned earlier, because a separate web interface and logic to connect to the database would have to be made.

3.1.7 Network Search Service

The network search service is made to verify the system as a communication carrier between a fixed and remote environment, and to show the modularity of the service platform. The network search is itself a useful, real world service. Equally important, is the verification of the system as a whole. The network search service is a validation and test tool used to highlight issues and give real examples presenting a fully working service platform.

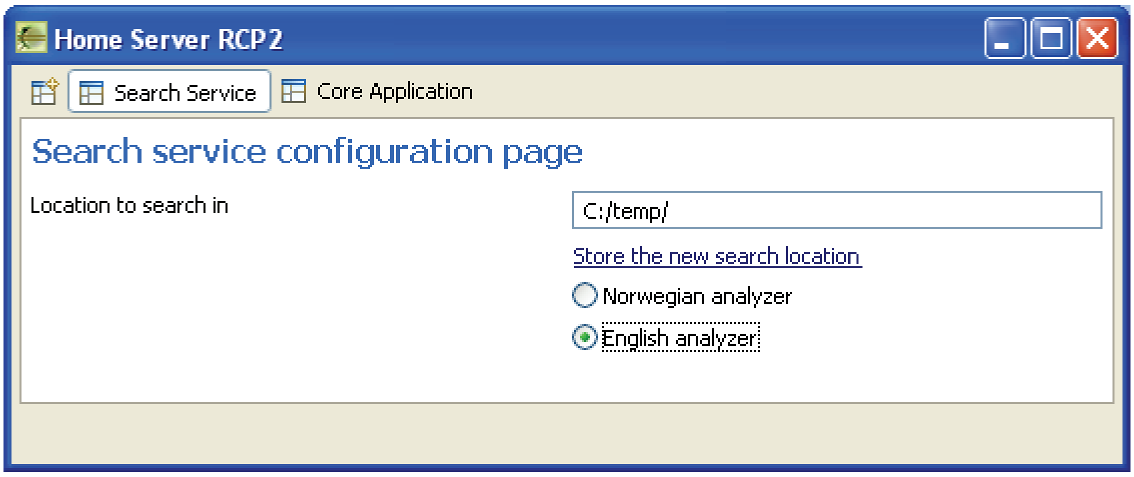
3.1.7.1 Architecture

The search service is a plug-in to the HS, enabling a user to search and download files on all available computers in the group, which have HS installed together with the search service.

It is built on Lucene, making both searches in filename, metadata, and content in text based mime types available to the user. The core of the search service is based on Lucene's powerful information retrieval and analyzing mechanisms.

The network search service is managed from the HS, and all the files to be indexed are stored on the computer where HS is deployed. Figure 31 shows the HS GUI after the search service is selected, and there are two parameters to configure:

- The folder to index
- The language of the documents in the index (gives more accurate hits)



Search service configuration page

Location to search in: C:/temp/

Store the new search location

( ) Norwegian analyzer
(●) English analyzer

Figure 31 - HS view of the search service

At the web client, the query string is typed and hits are shown and the files can be downloaded through the XMPP client at the NS. The hits are displayed to the user through the web page, and the visual interface is based on Google search. Most Internet users are familiar with Google search and its interface, and by displaying the hits as Google search, the usability and user experience increases. The service can be divided in two modules; one search and one file transfer module.

3.1.7.2 Search

A high-performance and flexible method was needed to make an information retrieval system running on top of the system architecture. Several solutions were considered, including:

- JCIFS library
- Google desktop
- Windows search 4.0
- Lucene library
- Xapian library
- Egothor library

JCIFS has little to do with information retrieval, but is a Java library allowing access to the SMB/CIFS protocol. It gives access to all resources available on the computer, mainly Windows machines, but also Linux machines using Samba and Open Solaris from the Nevada builds 84 and later [127]. JCIFS have functionality to access resources available to the computer, but it has no logic meeting the requirements of an information retrieval system, therefore all the complex logic and mathematics must be built upon the file system access capabilities. Due to this, other methods were examined to meet our needs.

Corporate applications such as Google desktop [128] and Windows search 4.0 [129] allow third party applications to use the system APIs to provide search functionalities. Both applications support metadata and content based search, but if the search service was designed to use one of the applications as an information retrieval system, the application must have been installed separately in addition to the HS and the search service plug-in. This would give a more complex and less user friendly service, and the advantages of doing so compared to the following solution was more or less not present.

Apache Lucene [105], Xapian [130], and Egothor [131] are three full-text indexing and searching libraries with complex logic to support advanced information retrieval. All libraries are flexible enough to be used in the search service, and have performance matching the usability and user experience we wanted to provide the users. To differ between the libraries, the user group and documentation was examined closer. Lucene had the most active user group and the library with most and best documentation. There were also many sub projects linked to Lucene, i.e. Apache Tika. Based on this, it was decided to build the search architecture using Lucene as a fundament to the search service.

Lucene needs a folder to index and a search query as inputs. The output of Lucene are hits between the index and the query string, these hits are ranked based on the document score.

To provide the user with as accurate hits as possible, the analyzer must be modified to meet our needs.

- Stemming, stop words, support for multiple mime type, and multiple language support are features built into the search service to give the user a better accuracy and relevance based on the query.
- Summary and highlighting are features built into the search service to give the user a better experience when the hits are displayed.

3.1.7.3 File transfer

MSN web messenger [132] and Google talk [133] are examples of corporate applications with no support for file transfer. Google talk supports voice and video if a module is downloaded and installed. The service created in this thesis must provide file transfer without:

- Any type of installation to the browser or file system
- Any type of user interaction to other systems than those integrated in the platform
- Any technologies unavailable to native browsers

Due to the choice of using a web portal as the interface towards the remote device, file transfer is difficult to achieve without the user installing or configuring other systems. Problems such as: Same origin, JavaScript restrictions, and NAT made it difficult to achieve a service allowing file transfer. The same origin prohibits content from another location than the web server to be downloaded in order to protect the user. JavaScript does not allow a web application to access file I/O using native browser capabilities. Most NATs prevent the web client to access the HS from outside the NAT directly, making a URL sent from the HS to the web client with the path to the file of no value. To get around the NAT problem, the router must enable port forwarding, and the HS must have a web server integrated to deploy the file to a web page. Solutions making the user interact with other systems than ours makes the system less user friendly, therefore another solution had to be found.

The Meebo IM project [134] was used as an inspiration to solve problems related to file transfer. Meebo provides file transfer between two web clients, no installation, no port forwarding or other interactions to other systems are needed, and it uses only native browser capabilities. Meebo makes use of the Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3) [135] to provide their users with file transfer through a web based IM network. We used the same architectural thoughts when we designed the file transfer incorporated in the search service. Instead of connecting to the Amazon web services, it was decided to let the NS act as storage of files.

When a file transfer is requested from the web client, the HS has to use a third party to provide the file to the web client, just as in Meebo. Our solution was to develop an XMPP client, similar to the HS, deployed at the NS. The NS should function as “the cloud”, and offer the file to the web client. The NS client has a connection to the XMPP server and has access to the file system of the Network Server through Java. When the NS client has stored the file, received from HS, the web client is informed and the file is available to download through the web interface. In Figure 32, an overview of the file transfer solution is shown.

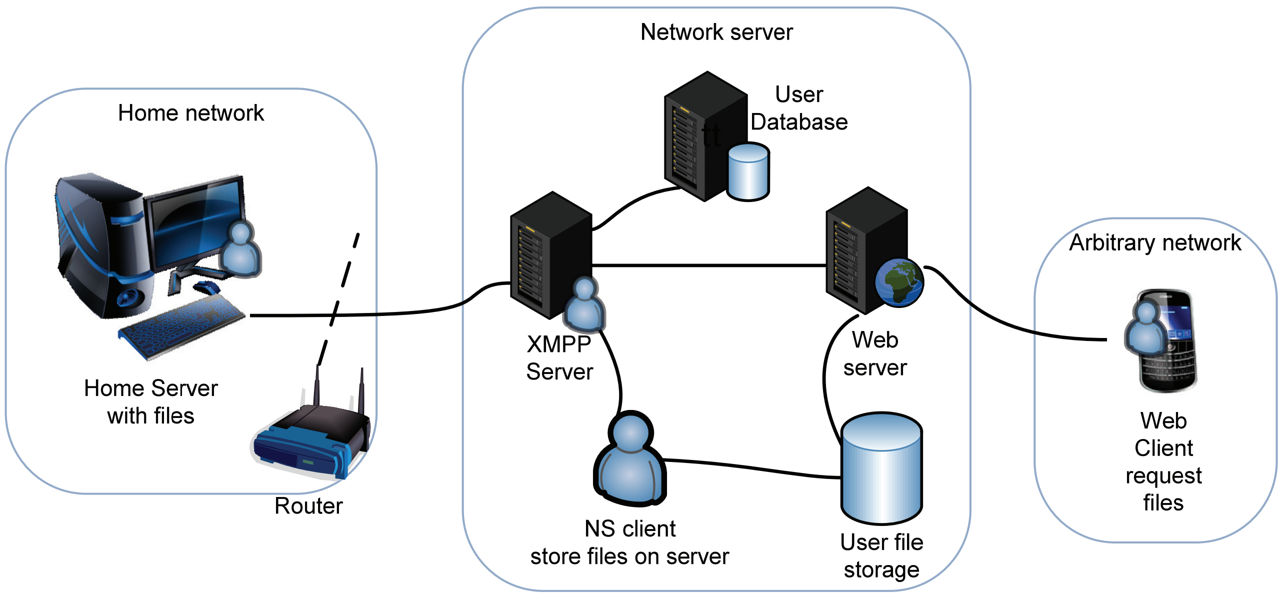


Figure 32 - File Transfer solution with NSclient and user storage

The integration of the NS client to the system can be used by other services in future development.

3.1.7.4 The service plug-ins to the Home Server

As described in the previous paragraphs, the search service should be added as a plug-in to the service platform. This is performed by creating a set of plug-ins which can be loaded into HS at runtime. Appendix A – Plug-in development for the Home provides a tutorial describing how to develop plug-ins to this application. The following is a short description of the naming scheme and content of each plug-in. Which functionality to include in each plug-in is essentially a design choice, hence it is included here.

The Search Service: no.telenor.services.search

The search plug-in, which is developed as an Eclipse plug-in is an index and search application. It will provide the Web Client with the possibility to search a pre-defined location on the Home server containing the plug-in.

The search will be powered by Apache Lucene v.2.4.1 and Apache Tika v.0.3 which enables full text search of a wide variety of file types. The query will be returned to the user together with the ranked results and optionally the content of the file, with highlights of the search query.

The Lucene search API: no.telenor.services.search.apache.lucene

This plug-in provides the search service with the Apache Lucene library files necessary for utilizing the Lucene built in features. Apache Lucene is a powerful open source search and index engine described in earlier chapters. This plug-in does not contain any code other than the Lucene libraries.

The Tika content API: org.apache.tika.no-sax

The Tika Plug-in includes the library from the Apache Tika Project.

3.1.8 Design summary

The design chapter provided an overview of the system and its design. The high level design choices have been discussed without going into the actual code. The main entities in the system, the Home server, the network server and the web client have been described. In addition, the modular structure of the Home server has been discussed as well as the service made to show the use of the system. This is done to highlight the importance of maintaining a separation between the service platform and the actual services utilizing this platform.

3.2 Implementation

The discussions in the previous design chapter and the technologies described in the theory part of this thesis have been on a high level and have mainly focused on selecting the right technologies. XMPP was selected as an open standard with the right properties; XML was selected to provide an extensible way to exchange information. Eclipse RCP, was selected as a modular platform to host services in the home network. The Openfire server from Jive software was selected as the XMPP server and so on. This chapter will discuss how these technologies were implemented in the context of the Network Server and the Home Server. The discussions will consider the implementation details and choices made in each of the technologies chosen in the design chapter.

3.2.1 Network Server, NS

The Network Server is the provider of the web portal, and a provider of connectivity between the remote device and the home network. The NS in itself is only a development server which should meet the requirements of the connectivity and the web portal. It was decided to focus on free software with as little configuration needs as possible to contain focus on the main goal. All the entities installed on the NS are free commercial products, except the XMPP client which is developed in this project to suit our needs. The rest of this chapter is mostly devoted to the web portal and related subjects, in addition some sections cover the commercial software and other important aspects regarding this project.

3.2.1.1 Web server

Apache2 is available in Ubuntu and can be installed through the terminal using:

```
sudo apt-get install apache2
```

The configuration of Apache2 is performed by giving directives through textual configuration files, such as the main configuration file in apache: *apache2.conf*. Basic configuration is not explained in this report, trying to stick to the core of things. However, configuration details, concerning this project explicitly, are revealed. One such example is the proxy settings to our XMPP server. For more configuration information, take a look at [118] [119].

BOSH, the transport protocol between the remote device and the XMPP server

To connect the BOSH requests and responses between the XMPP server and the remote device, the web server must function as a proxy. The web server relays all BOSH packets to the XMPP server, where the packets are processed. To configure Apache2 as a proxy, the Apache module *mod_proxy* [136] must be loaded. The following changes are made in apache2.conf, the configuration file in Apache2, to allow Apache2 to function as a proxy for BOSH messages:

```
LoadModule rewrite_module /usr/lib/apache2/modules/mod_rewrite.so

<Virtualhost *:80>
      ServerName localhost
      DocumentRoot /home/pc1/www/
      <Directory /home/pc1/www/>
            Options +Indexes +MultiViews
      </Directory>
      AddDefaultCharset UTF-8
      ProxyRequests Off
      <Proxy *>
            Order allow,deny
            Allow from all
      </Proxy>
      ProxyPass /http-bind/ http://127.0.0.1:7070/http-bind/
</Virtualhost>
```

When a BOSH request is sent from the remote device to the web server, the rewrite module [137] in Apache2 determine that the appropriate server to handle this request is located at port 7070 on local host. I.e. it will be forwarded to the XMPP server.

3.2.1.2 The web portal

The web portal uses web technologies available to most browsers. The basic building blocks of the web portal are HTML, CSS, and JavaScript; these technologies are supported by most desktop and later mobile browsers. To avoid all the low level coding in XMPP, a JavaScript library called JSJaC [138] was used. JSJaC is a XMPP JavaScript library making the development of a web based XMPP client easier and faster. JSJaC is flexible and have the functionality needed to be used in this project as an XMPP library.

The structure of the web portal is built up by three folders, core, services, and application as shown in Figure 33.

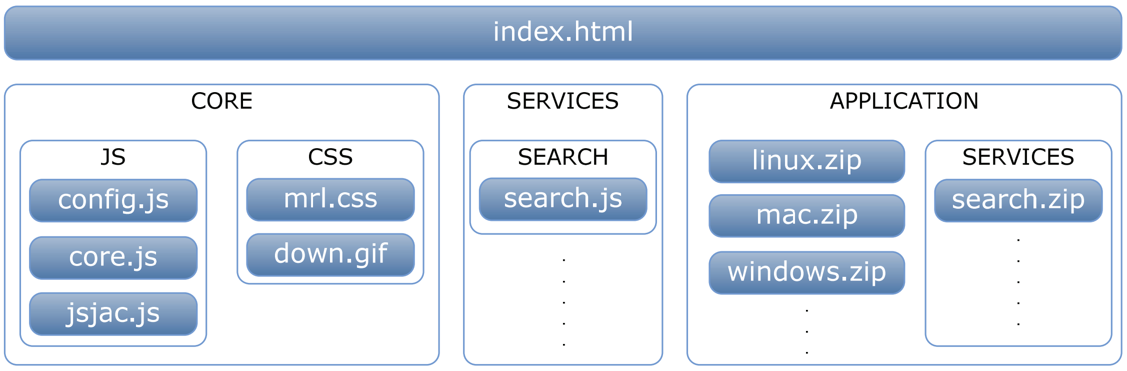


Figure 33 - Architecture of the public directory on the web server

The JS folder contains the core JavaScripts to complete connectivity to the XMPP server, and the CSS folder contains style information of the portal. The last two folders are used to deploy new services and application releases. How developers can make new services and application releases available to the users through the web portal is described in 3.2.1.5.

The core.js defines a set of available global variables which can be used to access information about the web client:

- globals.useHS
- globals.listOfHS
- globals.availableHS
- globals.unavailableHS
- globals.presenceAvailable
- globals.HSRoster
- globals.webClientRoster
- globals.myJID

Globals.useHS contain the name of HS which is currently used, listOfHS contains an array with all HS stored in the roster. Globals.availableHS and unavailableHS are arrays storing presence information about all the HSs in the roster, the presenceAvailable array store all available clients; NS clients, web clients, and HSs. Globals.HSRoster and globals.webClientRoster are arrays storing the rosters of respectively the connected HS, the one in globals.useHS, and the roster used by the web client itself. The globals.myJID stores the information of the username of the connected user.

Config.js is a configuration file to edit the most common parameters available in JSJaC, the most important parameters are:

- Communication protocol
- Domain
- Authentication type
- Credentials

Config.js also extracts the username and password from the login form available to the user. The code on the next page shows the doLogin function in the config.js file. The function is invoked when the user tries to log in.

```
function doLogin(aForm) {
        if (aForm.username.value != '' && aForm.password.value != ''){
                try {
                // setup args for contructor
                oArgs = new Object();
                oArgs.httpbase = "http://129.241.208.209/http-bind/"
                oArgs.timerval = 2000;
                if (typeof(oDbg) != 'undefined')
                        oArgs.oDbg = oDbg;
                        con = new JSJaCHttpBindingConnection(oArgs);
                        setupCon(con);

                        // setup args for connect method
                        oArgs = new Object();
                        oArgs.domain = "129.241.208.209";
                        oArgs.username = aForm.username.value;
                        oArgs.resource = 'MRL';
                        oArgs.pass = aForm.password.value;
                        oArgs.authtype = 'nonsasl';
                        con.connect(oArgs);
                } catch (e) {
                } finally {
                        return false;
                }
        }
}
```

The JID is the address of an XMPP entity, and has the following elements:

node@domain/resource, oArgs.username@oArgs.domain/oArgs.resource

The resource identifier is an optional tertiary identifier, but with different resource identifiers an entity can maintain multiple connected resources simultaneously [85]. This can be an option in the further development of the web portal, and the resource identifier is therefore available in the configuration file.

The oArgs.httpbase configures the type of connection the client should use to connect to the XMPP server. JSJaC supports the deprecated HTTP polling [139] and Bidirectional-streams Over Synchronous HTTP (BOSH) [91] [92].

Roster implications

When a new HS or web client is created from the HS, this must be handled by the web client. The web client must get access to the new HS through the web portal in order to access the services and the home network. If a new web client is created, all existing clients should also be able to store the new client in their roster. This is not important for the search service where a web client only communicates with HSs, but this feature could be used by future services. In order to configure the roster stored in the XMPP server and both incoming and outgoing subscriptions, a set of messages defined in RFC 3921 [86] must be handled.

If the web clients do not send roster management packets to the server, roster management is handled by the XMPP server. Both the web client and the Home Server must send roster management packets in order to create the group name, which is used to filter different types of clients.

The next sections show how roster management should be handled by an XMPP client. First a case showing what happens when the web client initiates a subscription, and the second case shows how the web client behaves when a subscription request is received. The two usernames web1 and hs1, respectably the web based client and the Home Server, is used throughout the following example to show the roster management.

In the case of adding a new user to the roster, an IQ packet to the server is sent to prepare the server and keep track of the subscription.

```
<iq type='set' id='set1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='hs1@example.org' name='nickname'>
      <group>HS</group>
    </item>
  </query>
</iq>
```

The subscription is sent after the roster IQ packet to request a subscription, in this case to *hs1@example.org*.

```
<presence to='hs1@example.org' type='subscribe'/>
```

After sending the *subscribe* message, the web client should get a feedback of either *unsubscribed* or *subscribed*. If *unsubscribed* is received the web client respond sending *unsubscribe*, and the web client is not stored in the *hs1* contacts roster and have subscription state *none* in the web client roster.

```
<presence to='hs1@example.org' type='unsubscribe'/>
```

If the presence stanza received is *subscribe*, the web client sends a confirmation of the subscription by sending a new *subscribe* message to *hs1*. This is an optional presences message and notifies the server it must no longer send subscription changes to *hs1*; this should not affect the subscription state. The web client now has access to the *hs1* presence information indicated with presence stanza *to*, and *hs1* have a subscription from the web client.

```
<presence to='hs1@example.org' type='subscribe'/>
```

In the second case, the web client receives a subscription request and responds with a *subscribed* or *unsubscribed*. If the request is from a valid HS client, the web client responds with a roster set to the XMPP server and a *subscribed* message to the *hs1*.

```
<iq type='set' id='set2'>
  <query xmlns='jabber:iq:roster'>
    <item jid='hs1@example.org' name='nickname'>
      <group>HS</group>
    </item>
  </query>
</iq>
```

```
<presence to='hs1@example.org' type='subscribed'/>
```

After the *subscribed* message is sent, the web client has a subscription from the HS and a presence stanza *from* to the *hs1* but does not have access to *hs1* presence information. The next step for the web client is to make a mutual subscription to access the *hs1* presence information. To create mutual subscription, the web client has to send a *subscribe* message to *hs1*.

```
<presence to='hs1@example.org' type='subscribe'/>
```

After the message is sent, a response with presence stanza *subscribed* from *hs1* is received and mutual subscription is achieved between *hs1* and the web client. The web client presence information to *hs1* is changed from *from* to *both*, and the web client has access to *hs1* presence information. To notify the server it must no longer send subscription changes to *hs1,* the web client sends a confirmation of the subscription by sending a new *subscribe* message to *hs1*. This message should not affect the subscription state.

```
<presence to='hs1@example.org' type='subscribe'/>
```

Visual web interface

The web interface accessed by the web clients should be accessible from a wide variety of devices and browsers. At the moment no optimization is preformed to provide devices with small screens or other restrictions a better usability. This can be done by detecting the device connecting to the web page using CSS media metadata tags such as screen, mobile, etc., but this has not been implemented. Due to this, from a system point of view, the type of device connecting to the system does not matter. What does matter is the usability, and the user experience is better if the service is accessed using a device with a big screen and other human interface devices with good ergo metric support.

The systems web page has a simple and plain look, and has a minimalistic view. The page is divided into several sections to dynamically update just parts of the page, while the rest of the page is static. In the head section, Telenor and the project name is displayed. Between the head and main section, a download bar is displayed. This section shows all available downloads, both the desktop applications and plug-ins to the desktop application. The main section of the page is divided in two.

The left section list different choices available to the user, such as: available services, available and unavailable HSs, but it can also be used to show service specific information. If the left section is used to service specific information, the information must appear below the available and unavailable HSs section.

The right section holds information about status, service specific tasks, and other user interactions available such as log in. The status information is always located at the top, and changes dynamically depending on the user interaction. If a service is chosen from the left section, information and interaction option appears in the right section. A series of pictures show how the page dynamically updates when different actions occur. Figure 34 shows the login form available to the user, after the web page is downloaded by the browser.

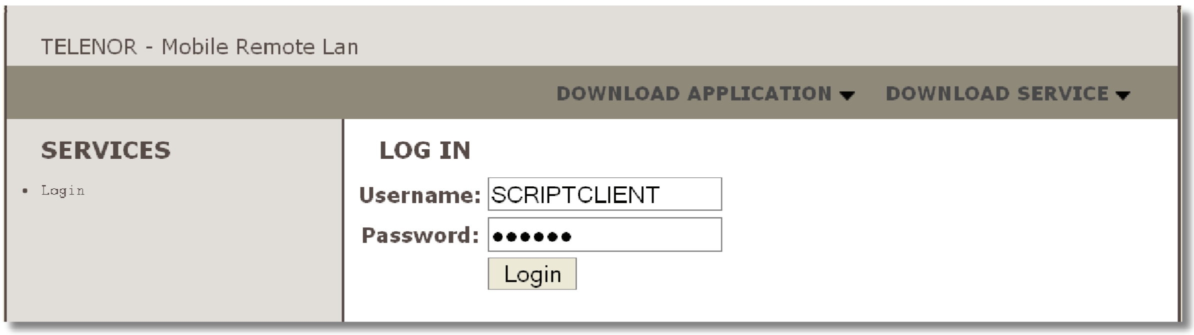


Figure 34 - Visual web interface, screenshot 1

If the user has a username and password, he can log in. If the user does not have a username and password, the only available options are located at the download bar. Figure 35 and Figure 36 show downloads available to the user.

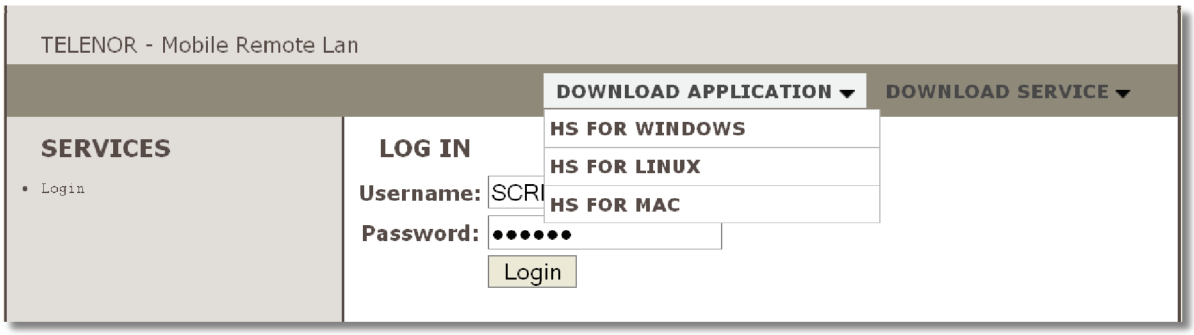


Figure 35 - Visual web interface, screenshot 2

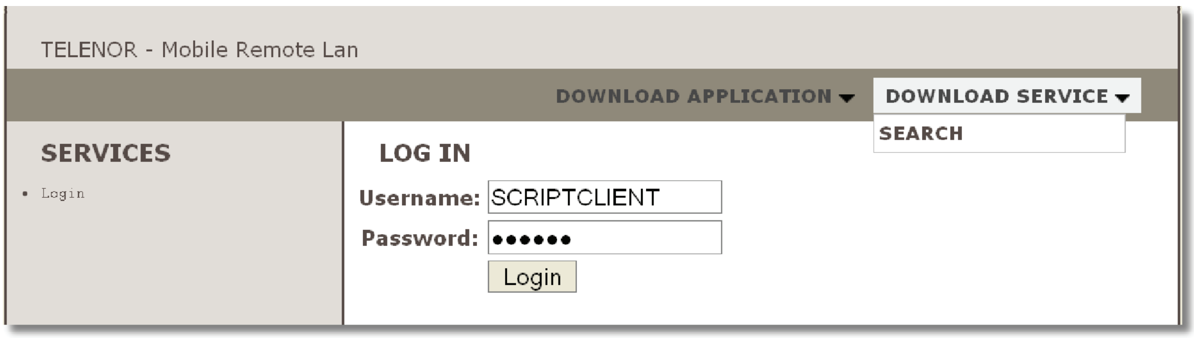
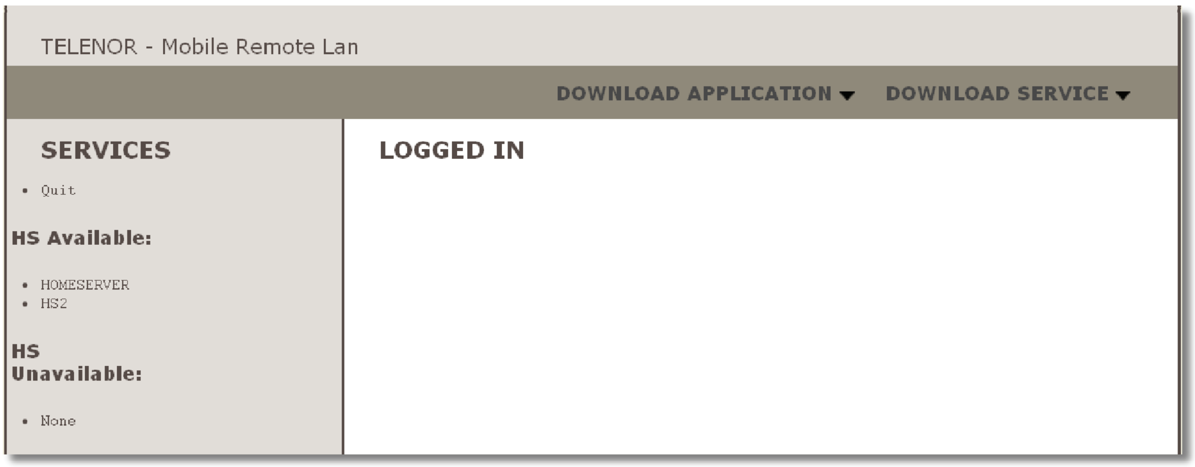


Figure 36 - Visual web interface, screenshot 3

If the user log in, shown in Figure 37, the left bar changes and the available and unavailable HSs are displayed. Also the status information located at the top right section changes from log in to logged in.


TELENOR - Mobile Remote Lan

DOWNLOAD APPLICATION ▾ DOWNLOAD SERVICE ▾

SERVICES
- Quit

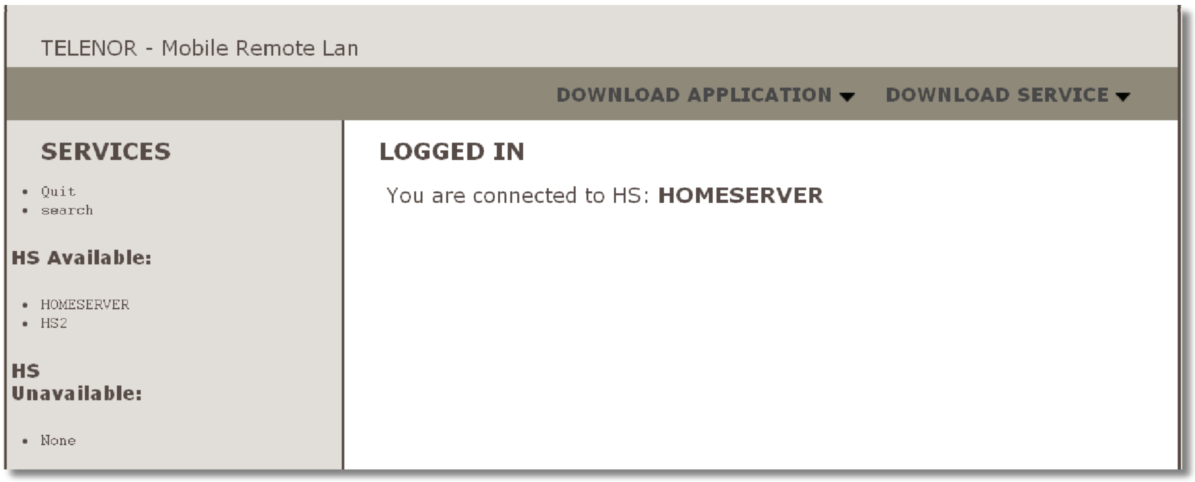
HS Available:
- HOMESERVER
- HS2

HS Unavailable:
- None

LOGGED IN

Figure 37 - Visual web interface, screenshot 4

The user have now two choices, either log out using the service "Quit" or connect to a available HS. If the user connects to an available HS, the left content is updated with available services, and the right content gives the user information of which HS he is connected to, see Figure 38.


TELENOR - Mobile Remote Lan

DOWNLOAD APPLICATION ▾ DOWNLOAD SERVICE ▾

SERVICES
- Quit
- search

HS Available:
- HOMESERVER
- HS2

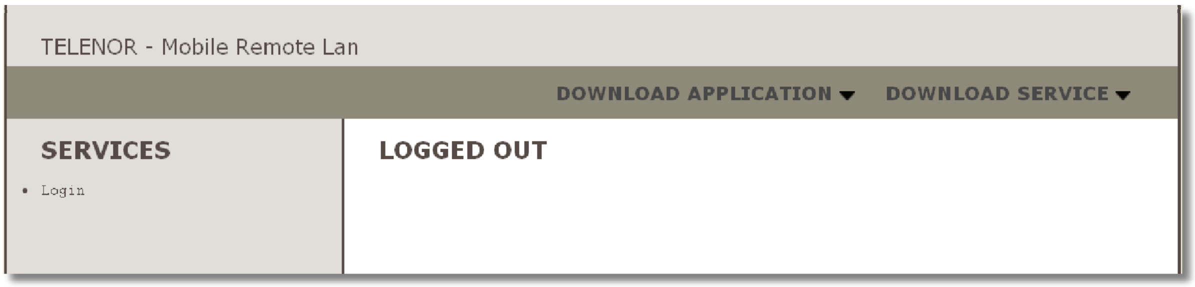
HS Unavailable:
- None

LOGGED IN

You are connected to HS: **HOMESERVER**

Figure 38 - Visual web interface, screenshot 5

If the user decides to log out, the left content changes and shows only the login service, while the right content shows the new user status, see Figure 39.


TELENOR - Mobile Remote Lan

DOWNLOAD APPLICATION ▾ DOWNLOAD SERVICE ▾

SERVICES
- Login

LOGGED OUT

Figure 39 - Visual web interface, screenshot 6

At the moment the system is tested and supports the following browsers:

- Internet Explorer 8
- Firefox 3.0
- Opera 9.0
- Google Chrome 1.0
- Safari 3.2
- Mobile Safari with iPhone OS 2.2
- Opera Mobile 9.5 beta

3.2.1.3 Search service

The search service is implemented in JavaScript, using the core of the web portal to handle messages to and from the HS application. The protocol tag: MessageType is *search* and the Type tag has three possible values:

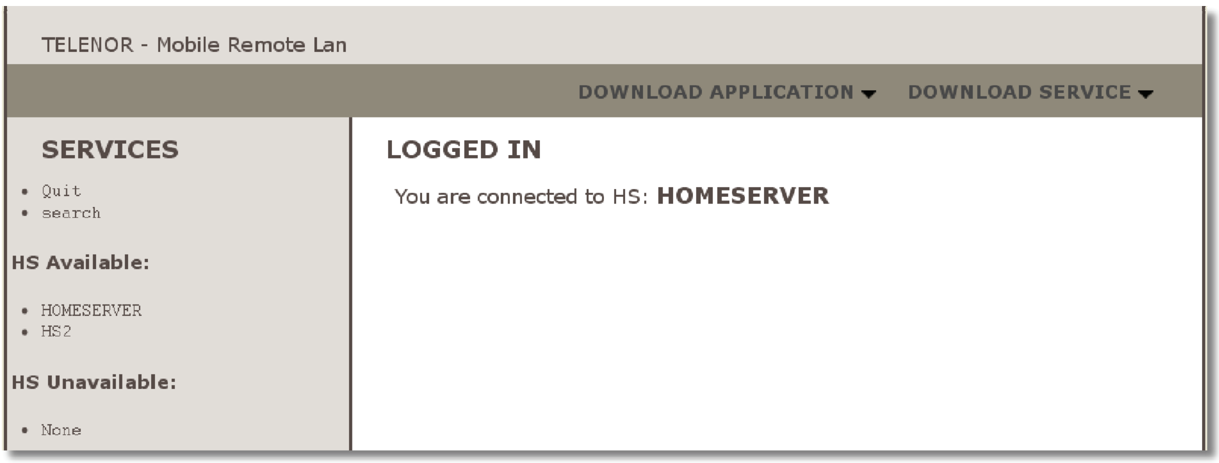
- Response
- UserFiles
- service-unavailable(503)

If no type is specified, or another type is received, the Type tag is displayed to the user together with information of the sender of the message packet.

When the search service is selected, a form where the query string can be typed is presented to the user. In addition to the text field, a checkbox is used to decide where to send the query. The checkbox is checked by default, and the message is sent to all available HSs. If the checkbox is unchecked, the query is sent to the connected HS.

3.2.1.4 Visual web interface, search service

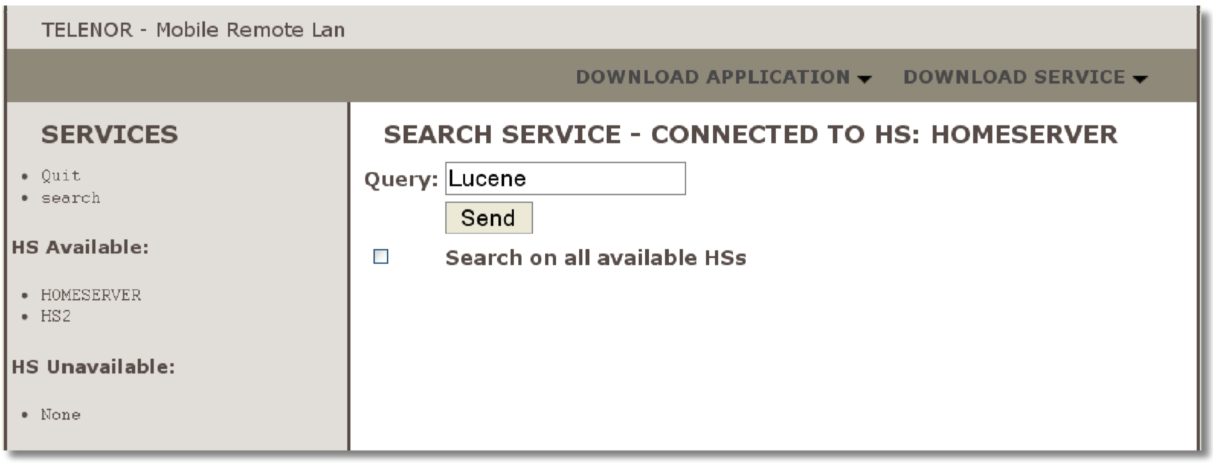
The GUI on the web client must be easy and self explaining. This section shows how the search service looks to the user, and explains the user interactions in three simple steps.



You are connected to HS: **HOMESERVER**

Figure 40 - How to use the search service, step 1

After the HS is selected, the search service should appear below the services header in the upper left corner, see Figure 40. When the search service is accessed, the main window is updated and the capabilities of the service are shown, see Figure 41.


TELENOR - Mobile Remote Lan

DOWNLOAD APPLICATION ▾ DOWNLOAD SERVICE ▾

SERVICES

- Quit
- search

HS Available:

- HOMESERVER
- HS2

HS Unavailable:

- None

SEARCH SERVICE - CONNECTED TO HS: HOMESERVER

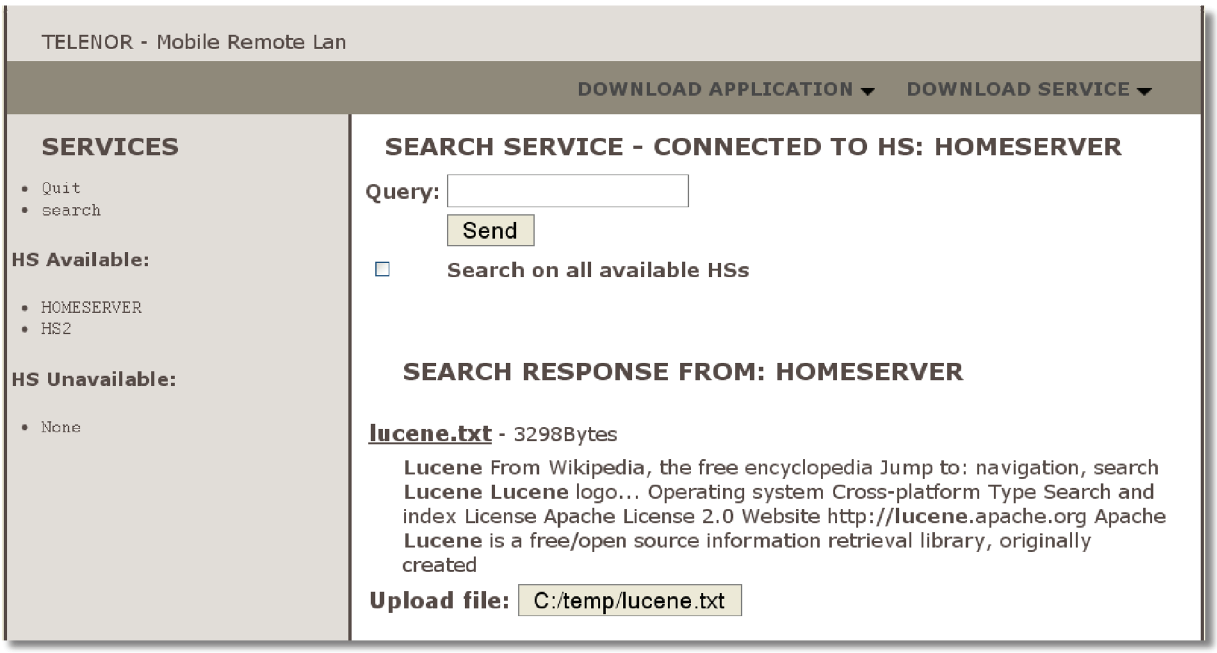
Query: Lucene

Send

☐ Search on all available HSs

Figure 41 - How to use the search service, step 2

The search service have only to fields available to the user to configure; the query string and the choice of searching for the file in all available HSs or just the one which is connected at the moment. By typing "Lucene" as the query and not check the "search on all available HSs" button, the following result is displayed on the screen, see Figure 42.


TELENOR - Mobile Remote Lan

DOWNLOAD APPLICATION ▾ DOWNLOAD SERVICE ▾

SERVICES

- Quit
- search

HS Available:

- HOMESERVER
- HS2

HS Unavailable:

- None

SEARCH SERVICE - CONNECTED TO HS: HOMESERVER

Query:

Send

☐ Search on all available HSs

SEARCH RESPONSE FROM: HOMESERVER

lucene.txt - 3298Bytes

Lucene From Wikipedia, the free encyclopedia Jump to: navigation, search **Lucene Lucene** logo... Operating system Cross-platform Type Search and index License Apache License 2.0 Website http://**lucene**.apache.org Apache **Lucene** is a free/open source information retrieval library, originally created

Upload file: C:/temp/lucene.txt

Figure 42 - How to use the search service, step 3

3.2.1.5 *How to add new services*

If a new service is to be added on the web server, there are a few steps the developer must consider:

- The name of the service, and the function first invoked when the services are accessed, must have the same name (see example below)
- Use the JSJaC library to handle received packets

- Add a new case in config.js with a MessageType and Types according to the protocol
- Include all the JavaScript files to core
- Add all the source code to the services folder located at the public folder on the web server

A list of available services on the HS is sent to the web client when the client connects to the respective HS. The message updates the service list in the left section in the web portal, and work as a link which can be accessed. If one of the services is accessed, the respective service should then be displayed in the right section of the web portal. To achieve this, the name of the service and the JavaScripts entry point in the service must have the same name. An example:

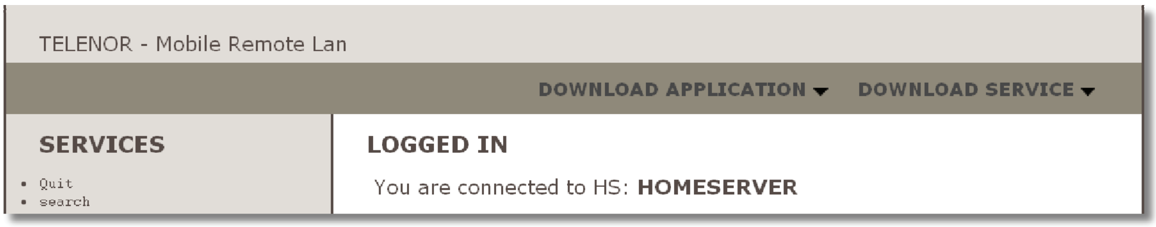


Figure 43 - Add a new service to the web server, example 1

In Figure 43, the services in the left section have a list containing a search service. If search is accessed, a function named search is invoked. Below is a snippet of the search function. This function is the entry point to the service, and the service name in the list and the function name in the JavaScript must be equal if the service shall be invoked.

```
function search(){
      var HS = globals.useHS.split("@");

      //main content
      var main = document.getElementById('main');
      while(main.hasChildNodes()) {
            main.removeChild(main.lastChild);
      }
      //sub main content
      var sub_main = document.getElementById('sub_main');
      while(sub_main.hasChildNodes()) {
            sub_main.removeChild(sub_main.lastChild);
      }
      //sub_left content
      var sub_left = document.getElementById('sub_left');
      while(sub_left.hasChildNodes()) {
            sub_left.removeChild(sub_left.lastChild);
      }
```

At the moment only the JSJaC library is supported by the web portal and therefore all manipulation of received packets must be used using this library. If another library is preferred to create and make XMPP packets, the library must be included in the service source files.

```
        // To add a service, a new case with a MessageType and Types
        // must be added according to the protocol
        switch(MessageType){
                case 'Communication':{
                        if(Type == 'Roster'){
                                parseHSRoster(xmlDocObject);
                                getWebClientRoster();
                        }else if(Type == 'Services'){
                                serviceUpdateMsg(xmlDocObject,
                                        aJSJaCPacket);
                        }
                        break;
                }
                case 'Search':{
                        if(Type == 'Response'){
                                displaySearchResponse(xmlDocObject,
                                        aJSJaCPacket);
                        }else if(Type == 'UserFiles'){
                                showFiles(xmlDocObject);
                        }else if(Type == 'service-unavailable(503)'){
                                showFilesUnavailiable(Type, aJSJaCPacket);
                        }else
                        break;
                }
                default:{
                        break;
                }
        }
}
```

The xmppListener function in config.js listen to incoming XMPP packets and must be configured to add a new service. A case to the switch statement in xmppListener must be added to allow handling of the new service packets. The packets must be created based on the protocol with MessageType and Type, the protocol is described in detail in the protocol section. A snippet of the xmppListener is revealed below, showing the core case with MessageType Communication, and search case with MessageType Search.

Before the service is available to the users, a folder with the source files must be included in the services folder at the public directory on the web server. The JavaScript files located in this folder must also be included in the core source code.

3.2.1.6 XMPP server, Openfire

Openfire is downloaded from "http://www.igniterealtime.org/downloads/" and installed. The setup procedure in Openfire is simple and basic settings for language, domain, database, profile, and administrator must be performed before the Openfire server is available to be used in the system. No more configuration than the external database is necessary to have the functionality needed in this project.

3.2.1.7 Database

The database built-in to Openfire supports the features we want. The use of the internal database however, does not support requests from anyone but the Openfire server. Therefore it was decided to use an external MySQL database.

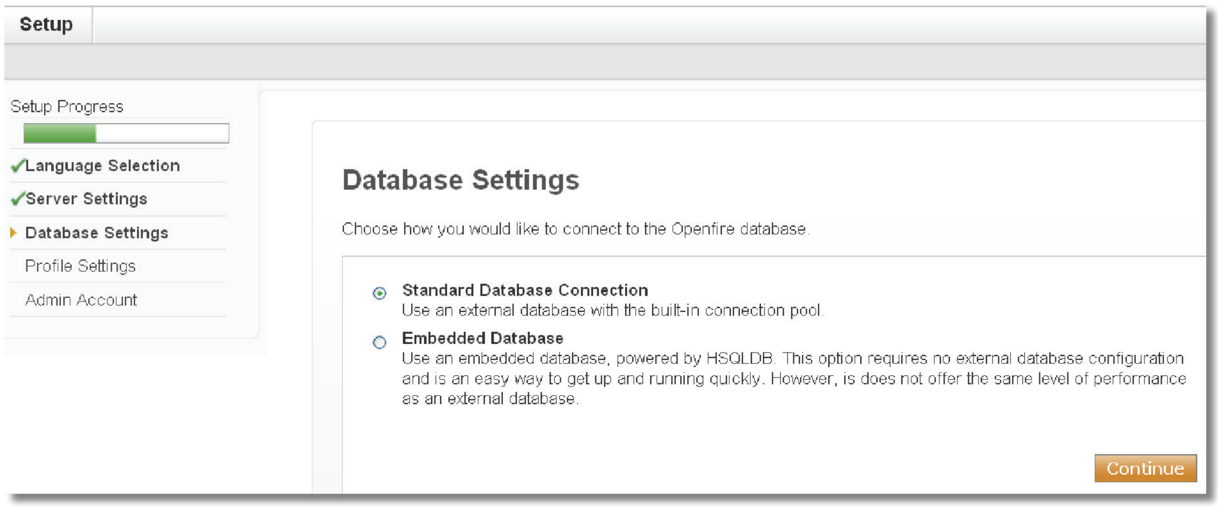
To use MySQL as the external database, the database schema must be copied to the external server and the database driver must be available to Openfire (the MySQL driver is preinstalled in the core on Openfire 3.6.4) before Openfire transfer the data. Before the data is transferred to the external database, MySQL is prepared by creating a database name using:

```
mysqladmin create Openfire
```

Openfire is the name of the database and can be changed to any preferred name. The next step is to copy the database schema from the Openfire server to the MySQL database named Openfire.

```
cat openfire_mysql.sql | mysql Openfire;
```

The MySQL database is now ready to accept data from the Openfire server. After the installation of the server the admin console is launched and the setup process begins where the external database can be configured. The database setup process is shown in Figure 44 and Figure 45.



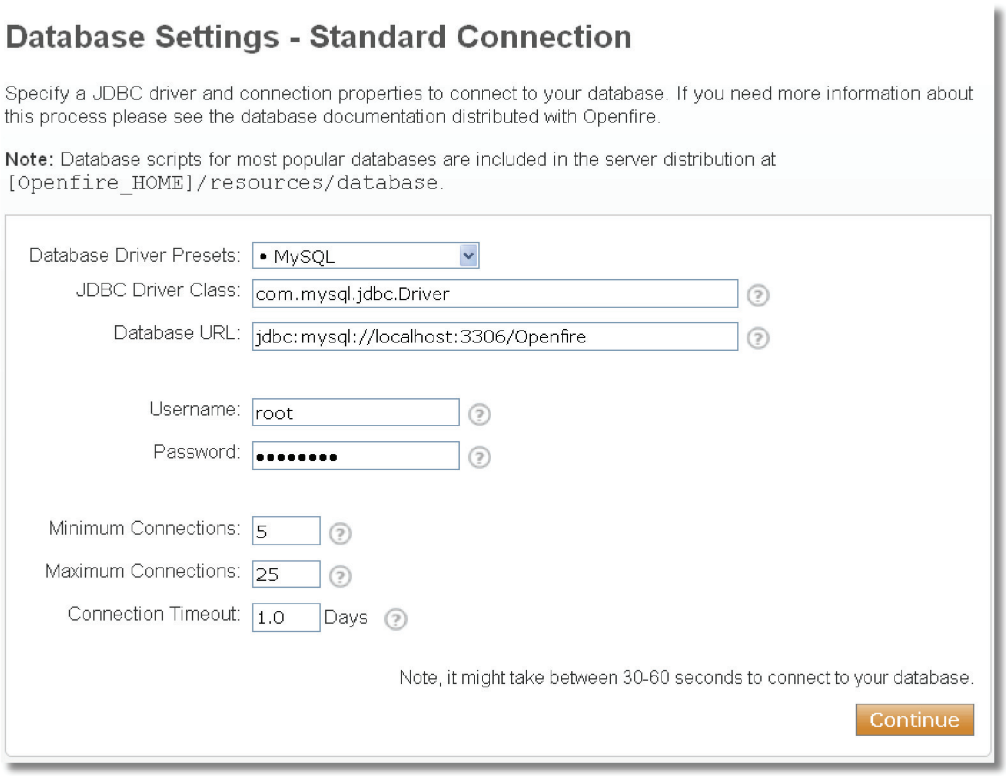
Database Settings

Choose how you would like to connect to the Openfire database.

[✓ Standard Database Connection]
Use an external database with the built-in connection pool.

Embedded Database
Use an embedded database, powered by HSQLDB. This option requires no external database configuration and is an easy way to get up and running quickly. However, is does not offer the same level of performance as an external database.

Figure 44 - Database settings in Openfire, screenshot 1 of 2



Database Settings - Standard Connection

Specify a JDBC driver and connection properties to connect to your database. If you need more information about this process please see the database documentation distributed with Openfire.

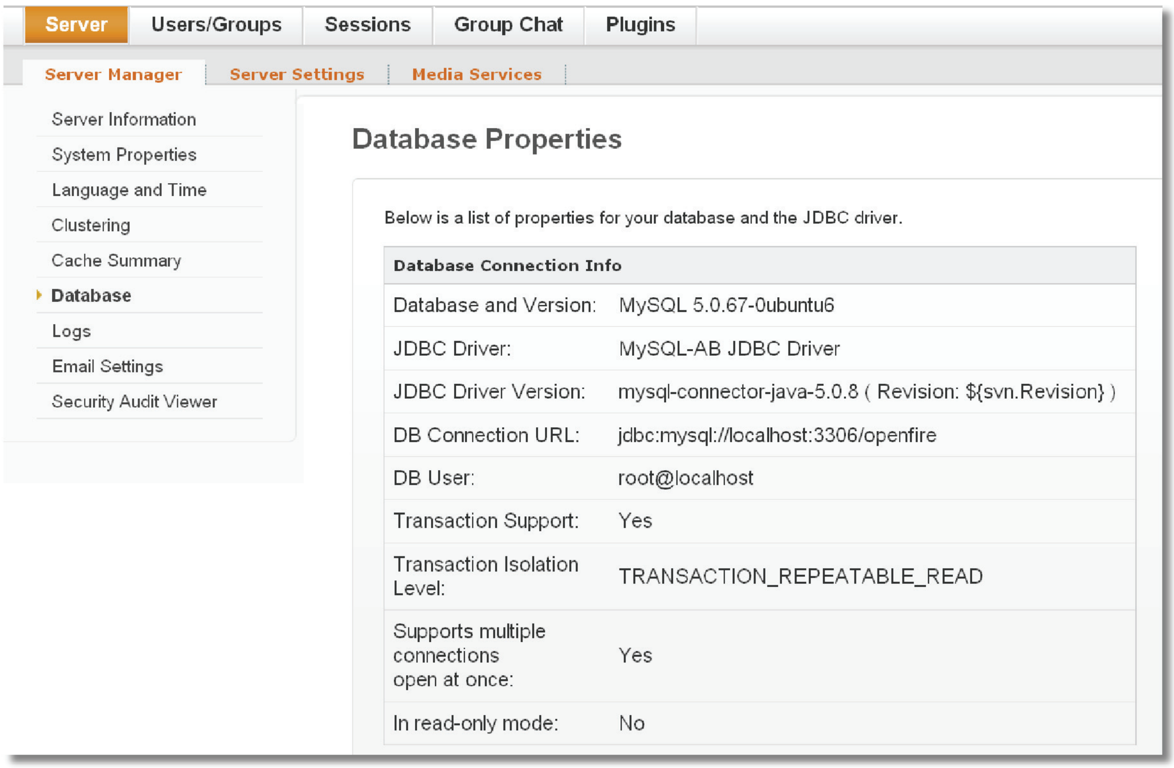
Note: Database scripts for most popular databases are included in the server distribution at `[Openfire_HOME]/resources/database`.

Database Driver Presets:	• MySQL
JDBC Driver Class:	com.mysql.jdbc.Driver
Database URL:	jdbc:mysql://localhost:3306/Openfire
Username:	root
Password:	••••••••
Minimum Connections:	5
Maximum Connections:	25
Connection Timeout:	1.0 Days

Note, it might take between 30-60 seconds to connect to your database.

Figure 45 - Database settings in Openfire, screenshot 2 of 2

After finishing the setup of the server, the database information can be accessed through the Openfire console and the traffic can be audited by Openfire to be used in statistics or debugging. Figure 46 shows the database properties on the Openfire server.



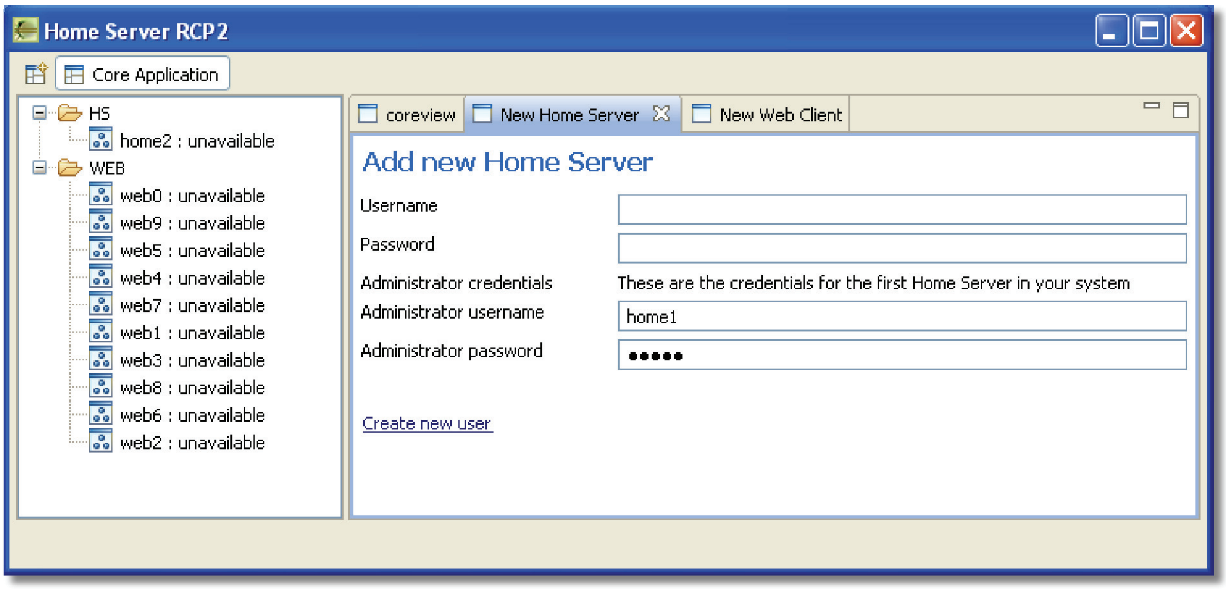
Database Properties

Below is a list of properties for your database and the JDBC driver.

Database Connection Info	
Database and Version:	MySQL 5.0.67-0ubuntu6
JDBC Driver:	MySQL-AB JDBC Driver
JDBC Driver Version:	mysql-connector-java-5.0.8 (Revision: ${svn.Revision})
DB Connection URL:	jdbc:mysql://localhost:3306/openfire
DB User:	root@localhost
Transaction Support:	Yes
Transaction Isolation Level:	TRANSACTION_REPEATABLE_READ
Supports multiple connections open at once:	Yes
In read-only mode:	No

Figure 46 - Database console view in Openfire

3.2.2 Home Server



HS
home2 : unavailable
WEB
web0 : unavailable
web9 : unavailable
web5 : unavailable
web4 : unavailable
web7 : unavailable
web1 : unavailable
web3 : unavailable
web8 : unavailable
web6 : unavailable
web2 : unavailable

Add new Home Server

Username	
Password	
Administrator credentials	These are the credentials for the first Home Server in your system
Administrator username	home1
Administrator password	●●●●●

Create new user

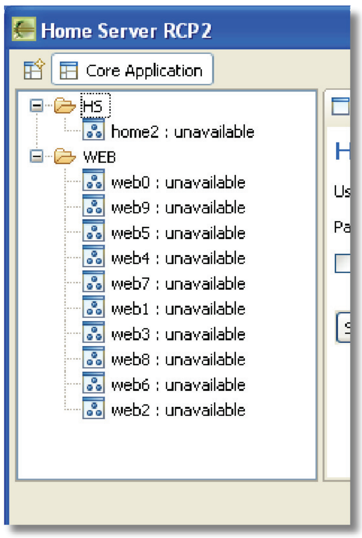
Figure 47 - The Home Server RCP application

The Home Server is the name of the Eclipse RCP client created and deployed on the desktop of the user's computer. The high-level system discussed in the design chapter will now be explained in more detail, going into the specifics of the modules of the system and their relationship. As mentioned earlier, the Home server application will consist of several parts, organized as plug-ins in an Eclipse RCP framework. The RCP framework is itself organized as plug-ins, which results in a hierarchy of plug-in dependencies.

The Hscore plug-in for example, rely on the org.eclipse.core.runtime.applications, org.eclipse.ui.perspectives, org.eclipse.core.runtime.products and the org.eclipse.ui.views from the eclipse RCP platform to be able to create a running application and user interface, while it relies on the no.telenor.smack plug-in to provide the functionality of the XMPP API made by JiveSoftware.

3.2.2.1 no.telenor.hscore plug-in

The core part of the application consists of all functionality responsible for the communication with the outside world and the major part of the user interface. Hscore provides the user with a basic GUI for setting the different parameters. Eclipse RCP organizes the GUI into Perspectives, a container filled with a custom number of views and editors. The Hscore perspective has four views.


Home Server RCP2

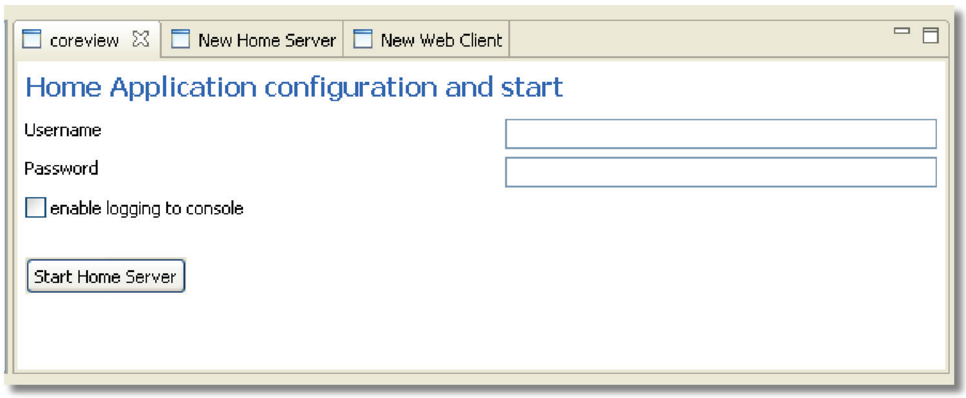
Core Application

HS
- home2 : unavailable

WEB
- web0 : unavailable
- web9 : unavailable
- web5 : unavailable
- web4 : unavailable
- web7 : unavailable
- web1 : unavailable
- web3 : unavailable
- web8 : unavailable
- web6 : unavailable
- web2 : unavailable

Figure 48 - The Roster View of the Home Application

The RosterView is responsible for showing a tree containing updated information about the various users with which this user has a relationship. This is very similar to the friends list in a chat client. This view is persistent throughout the other views in the plug-in. The three other views are organized in two different categories.


coreview | New Home Server | New Web Client

Home Application configuration and start

Username

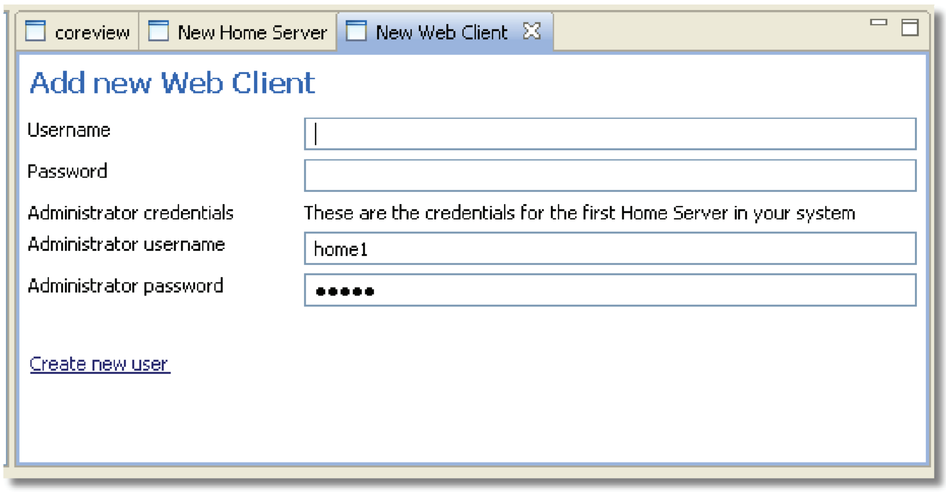
Password

enable logging to console

Start Home Server

Figure 49 - The core view of the Home Server

The CoreView basically starts the home server by logging in the user provided in the username and password fields. The WebClientView and HomeServerView are capable of creating new users.


coreview | New Home Server | New Web Client

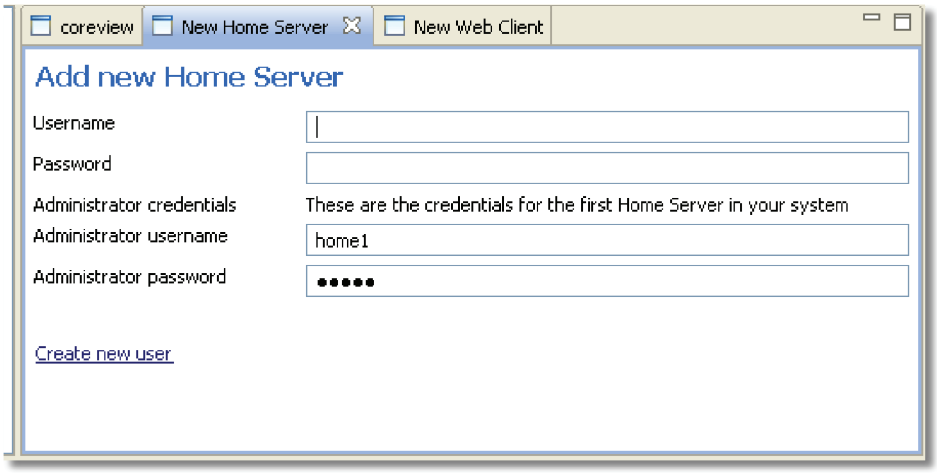
Add new Web Client

Username	
Password	
Administrator credentials	These are the credentials for the first Home Server in your system
Administrator username	home1
Administrator password	●●●●●

Create new user

Figure 50 - The New Web Client view

The WebClientView creates a user in the WEB group, associated with this home Server.


coreview | New Home Server | New Web Client

Add new Home Server

Username	
Password	
Administrator credentials	These are the credentials for the first Home Server in your system
Administrator username	home1
Administrator password	●●●●●

Create new user

Figure 51 - The New Home Server View

The HomeServerView creates new users in the HS group. These can either be self sufficient, or created in the group of this home server. The first home server created should be self sufficient, while the next ones should have an association with the first home server. This association is created by providing the username and password of the first home server. To avoid confusion this is called the administrator credentials, avoiding the need for the user to remember which account was his first user.

Figure 52 shows an overview of the Hscore plug-in. This figure is provided as an aid in understanding how the system is put together. As shown, all the Eclipse RCP classes are merged together as control classes. The GUI of the application is separated from the logic. The logic classes are merely POJO's, tied into the Eclipse RCP framework through the GUI classes.

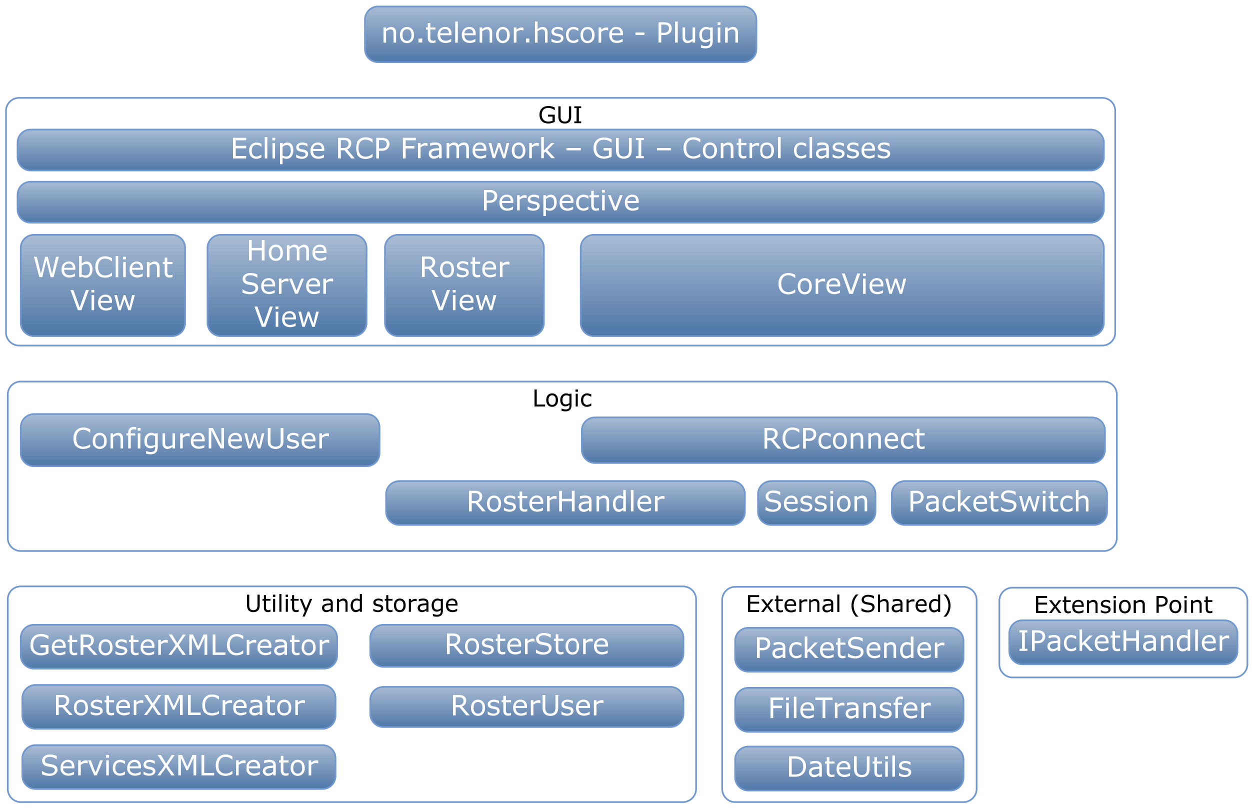


Figure 52 - no.telenor.hscore plug-in - Overview of the structure of classes and relationships

The most important classes are the RCPconnect, IPacketHandler and PacketSwitch. As illustrated in Figure 52, RCPconnect, which follows a singleton pattern, is initiated from the CoreView GUI. The RCPconnect initiates the XMPP connection and, with the help of the simple Session class, maintains it throughout the lifecycle of the application. It contains all the various event listeners and ties in with the utility classes to manage incoming and outgoing connections. It triggers on changes in the Roster and relays this to the RosterHandler, which process the change. Every time the roster is changed, an update is triggered in the RosterView GUI. When an incoming packet is received, RCPconnect relays it to the PacketSwitch, which decides whether it is addressed to the Hscore itself or if it should be relayed to the services through the IPacketHandler Extension point. All classes intended to be shared with other plug-ins are put in a package with external in its name. Eclipse RCP hides all packages from external use by default and any package intended for sharing with other plug-ins must be declared exported in the plugin.xml file.

3.2.2.2 no.telenor.hscore.org.jivesoftware.smack

The Smack API consists of four java libraries to establish an XMPP connection. The libraries are separated into four jar files; the smack jar contains the required parts of the XMPP protocol. The smackx jar file contains extensions to the protocol. Debug contains debugging classes and smackx-jingle contains the library necessary to enable peer to peer file transfer. The plug-in registers the no.telenor.smack extension point, which is extended by the hscore plug-in to enable use of the libraries. With the organization of the library as a standalone plug-in, it is possible to change the plug-in version without affecting the application itself. Figure 53 shows an overview of the smack plug-in

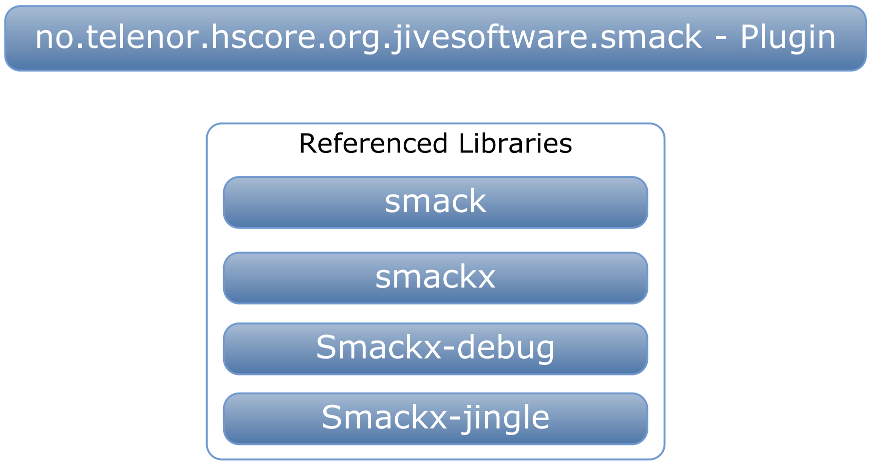


Figure 53 - The Smack API from Jivesoftware.org

3.2.3 Services

Services describe the modules which can be plugged into the core application to provide additional functionality. The core application itself does not provide much functionality valuable to the user; it provides a communication framework to connect several instances of the application with clients located in a remote location. The services gain access to this communication framework and can use them to provide additional functionality to the users. A plug-in developer's tutorial can be found in Appendix B. The tutorial covers the basics of creating a plug-in which takes advantage of the custom functionality found in the home application RCP. A broader understanding of the subject could be reached by reading one of the Eclipse-plug-in developer books available [140]. The next paragraphs will talk about the search plug-in which was created to showcase the functionality of the Eclipse RCP platform.

3.2.3.1 no.telenor.services.search plug-in

The purpose of the search plug-in is twofold. First it is a valid and useful implementation of an Apache Lucene content and metadata search engine. Second, it is included as an example of a service which can tie into the service platform created and maintained by the Hscore plug-in and the RCP application. The search plug-in contributes its own perspective to the GUI of the RCP application. Inside this perspective is a single view, with the possibility to alter the search location and the preferred language settings of the search. This illustrates how a service can be made customizable by the user. The service contributes an extension to the IPacketHandler extension point made available by the no.telenor.hscore plug-in. As described earlier, Hscore activates any plug-in extending the IPacketHandler extension point when it receives a packet with a service header. This makes the SearchPacketHandler class the focal point of the search service plug-in. It is through this class all indexing and search is executed. When a packet is received, it is interpreted, then the class checks if an index is created, optimizes it, and executes the specified search query on this index. It then sends a response packet back to the client by using an implementation of the FileSender class in the Hscore external package. Figure 54 shows a logical overview of the main parts of the plug-in.

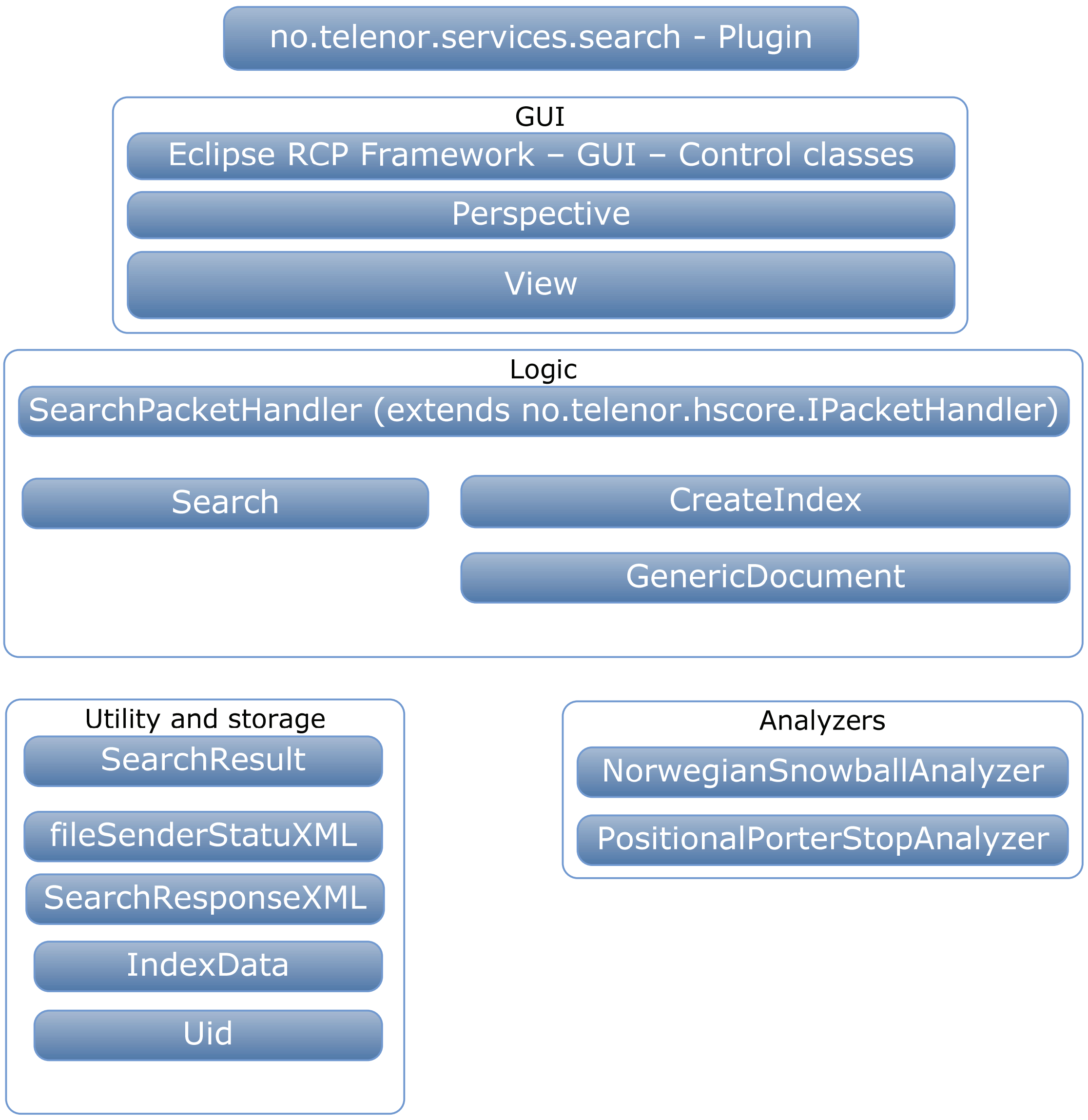

no.telenor.services.search - Plugin
GUI
Eclipse RCP Framework – GUI – Control classes
Perspective
View
Logic
SearchPacketHandler (extends no.telenor.hscore.IPacketHandler)
Search
CreateIndex
GenericDocument
Utility and storage
SearchResult
fileSenderStatuXML
SearchResponseXML
IndexData
Uid
Analyzers
NorwegianSnowballAnalyzer
PositionalPorterStopAnalyzer

Figure 54 - no.telenor.services.search -Plug-in an informal logical overview of the structure

Indexing

Whenever a call for the index is made, the application goes into the predefined folder where the index is stored. If an index is not present, one is created; otherwise the current index is optimized against the file system. The optimization process takes place by comparing the document name and last change data stored in the index with the documents in the file system. This name-change value is called UID and will act as a unique identifier for the document. If a document has changed since the last time it was indexed, it is deleted from the index and the new version is added.

When it comes to indexing, Lucene offers a large variety of options. All fields in the document can be indexed in different ways for different uses and optimizations. Because this implementation is capable of returning highlighted fragments of the search query, the full-text content of the document must be indexed. Additionally, the content must be analyzed for keywords. In addition to the contents of the document, the file name is also indexed and analyzed. To make the search for file names more accurate, the search implementation has chosen to boost the significance of the filename. The standard significance given to fields by default is one, by boosting the filename value with two; the score of the field is doubled. Consider a simple scenario where one document is called test.txt with "gibberish" content and another is named gibberish.txt, but has content "test" The document with the content test will receive a score of approximately half the value of the document test.txt. In other words, the test.txt file will turn up first in the search result even though they both contain one occurrence of "test". This functionality might be used to further customize the search to the user's needs.

Lucene offers the possibility to index documents based on language specific optimizations. It does this by taking away the stop words, e.g. and, or, a, etc. making the index smaller and faster. English and Norwegian analyzers are included as options for this specific project. The PositionalPorterStopAnalyzer and the SnowballAnalyzer is included for English and Norwegian respectively. Other languages are available from the Lucene project site.

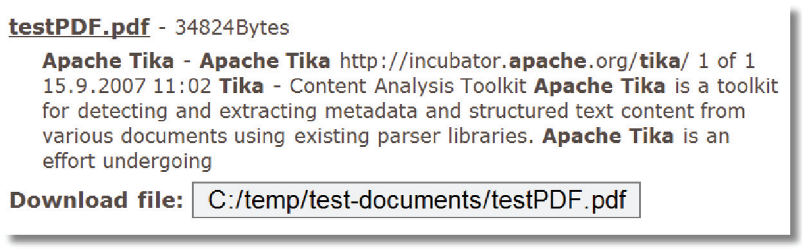
Indexing File Types

The search plug-in uses the Apache Lucene project to index the user files. This project does not provide parsers for different file types; it only includes a parser for text based files like .html and .txt. To be able to provide the user with a little more functionality, the "Apache Tika" project has been implemented. This enables the user to search in different file types, e.g. .doc, .docx, .pdf, .zip, .tar.gz, .mp3, .jpg and others, around twenty in total. A complete list can be found at the apache Tika web site.

To be able to index the content of a wide variety of file types is considered a major advantage for this project; it does however come at the cost of speed when building the index. Some of the file types, e.g. .docx, the native file type of Microsoft Word 2007, take quite a while to index. For this project it has been decided to index the content of all available file types to show the functionality of "Apache Tika". It would be beneficial to consider some logic to decide which file types to index in a future release of the search service. The user could have a choice to set up the indexer via the GUI. Some possible choices would be doc types, max size of the index, max size of document to be indexed and so forth. Another approach would be to let the search developer include some logic when it comes to speed, e.g. decided by the number of documents, the size of the indexed area or something else.

Searching

When performing a search with Lucene, it is executed on the index. The index is created the first time the search is performed. This may make the first search a bit slow. Once the index is created, it is only optimized for each search. This makes sure the search is up to date and relevant. Searching is performed on two parts of the document, the filename and the content. If the document content contains any of the search queries, the search returns three text-fragments of the file. The fragments are meant to give a hint as to what the document is about and highlights the query with a bold tag as shown in Figure 55.


testPDF.pdf - 34824Bytes

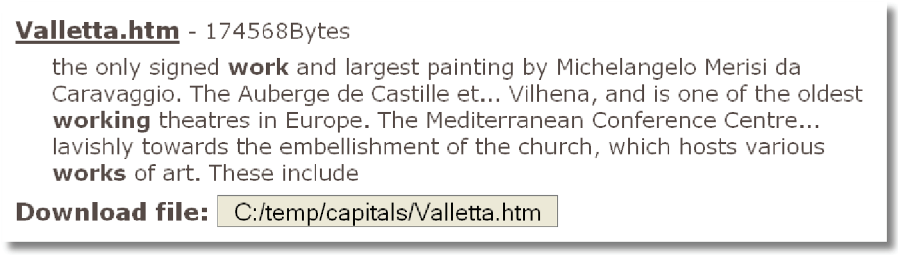
Apache Tika - Apache Tika http://incubator.apache.org/tika/ 1 of 1 15.9.2007 11:02 Tika - Content Analysis Toolkit Apache Tika is a toolkit for detecting and extracting metadata and structured text content from various documents using existing parser libraries. Apache Tika is an effort undergoing

Download file: C:/temp/test-documents/testPDF.pdf

Figure 55 - A typical search result with highlighting of the search query: *Apache Tika*

3.2.3.2 no.telenor.services.search.apache.lucene

The functionality utilized in the search plug-in is taken from the Apache Lucene API. To maintain modularity, this API is packaged as its own plug-in Figure 57 shows an overview of the jars included in the API. Lucene is organized in a core and a contrib. section. The contrib. section is optional and libraries from this section can be included when necessary. This setup is using two contrib. libraries, the highlighter and snowball functionality. The highlighter is used for highlighting relevant words and text fragments in the searched documents and returning these fragments to the client. The snowball is an optional analyzer to include support for other languages than English. The implementation uses the NorwegianSnowball analyzer as an option. Both analyzers are implemented with stemming, as shown in Figure 56 where the query “working” return and highlight the words work, works and working. This is a typical example of how stemming works.


Valletta.htm - 174568Bytes

the only signed **work** and largest painting by Michelangelo Merisi da Caravaggio. The Auberge de Castille et... Vilhena, and is one of the oldest **working** theatres in Europe. The Mediterranean Conference Centre... lavishly towards the embellishment of the church, which hosts various **works** of art. These include

Download file: C:/temp/capitals/Valletta.htm

Figure 56 - Query with stemming implemented: search word was "working"

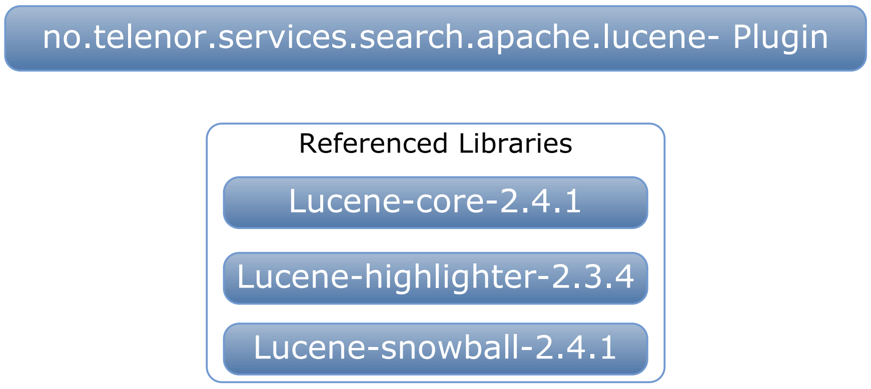


Figure 57 - Apache Lucene referenced library

3.2.3.3 no.telenor.org.apache.tika

The Library is built around Tika version 0.3, the version included in the appendix was compiled 2009-05-25. The library can be downloaded as source-only from the Tika web site, ready to build with Maven2 [141]. The developers use maven2 because it solves all dependencies necessary to run Tika. Maven2 downloads and includes all necessary dependencies upon compilation of the jar. Unfortunately for this project it also creates a conflict, which makes it necessary to alter the compiled Jar after it is made by maven2. It might also be possible to alter the maven2 configuration, but for the purpose of this project it was just as easy to do a workaround. Two versions are created as standard when compiling. One version contains only the Tika library, the other contains a standalone version of Tika containing all the dependencies.

The original thought was to use the standalone library unchanged. This caused a conflict because the library contained the org.xml.sax component, which is also present in the Sun Java 6 library in use by this project. When one of the methods in the project created an object containing an object of a class in the org.xml.sax java6 library and passed it to Tika, Tika received it and tried interpreting it with a different version of the org.xml.sax library. This conflict caused a crash. The workaround was achieved by removing the folder org.xml.sax from the tika-standalone.jar manually through the use of a standard archive editor, e.g. winRAR.

3.2.4 Protocol

Two scenarios are used to relay XMPP traffic over the network in this thesis.

- XMPP over TCP, between Home Server and Network Server
- BOSH: XMPP over HTTP over TCP between Web Client and Network Server

The first scenario uses the XMPP protocol directly on top of TCP, on the default port 5222. This is the most efficient, when considering overhead, and is the protocol of choice between the home application and the network server. The second scenario uses XMPP over HTTP over TCP, as described in BOSH – Bidirectional-streams over Synchronous HTTP. It is used when connecting the web client to the network server.

Some controversy exists in the choice of communication protocol for the different parts of the system. Since the system is meant to relay traffic over slow mobile links to the web client it would make sense to minimize overhead on this link. However, due to the limitations posed by using a web browser on the remote client, this is not possible and the less bandwidth efficient BOSH technology has to be used. The use of a web browser as a host for the XMPP client on the mobile client implicates the use of HTTP as the transport protocol. The current protocol, called BOSH – “Bidirectional streams over synchronous HTTP” is an XMPP extension to allow transport over HTTP [91]. The HTTP header in this case adds a substantial amount of overhead to the packet.

Even though the protocol adds extra overhead to the connection, it is substantially more efficient and provides lower latency than other similar protocols under the Ajax umbrella. BOSH achieves this low latency by using a technique called “long-polling” with multiple synchronous HTTP request/response pairs. BOSH is also fully compliant with constrained clients relying on HTTP 1.0 and without the use of cookies or access to HTTP headers [91]. In addition to this, the protocol adds additional application level reliability on top of the unreliable HTTP protocol [142].

3.2.5 Overhead

First let’s look at the overhead of the TCP/IP protocol stack.

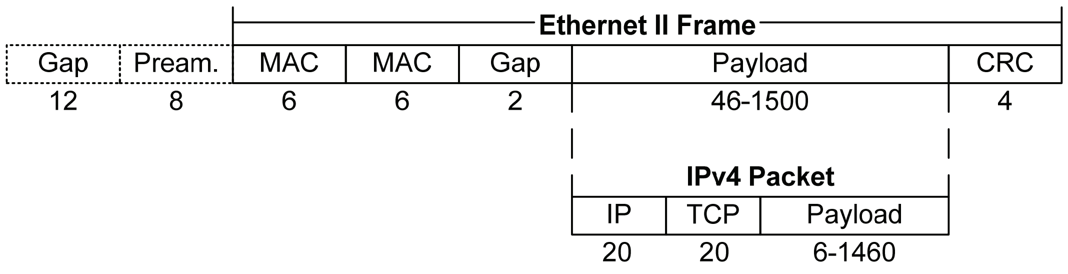


Figure 58 - TCP/IP Header and protocol [143]

As shown, the headers can be divided into the following categories.

- 18 bytes Ethernet header
- 20 bytes IP header
- 20 byte TCP header

Inside the payload of the TCP header, the application headers are distributed as shown here:

- 550 bytes approximate HTTP header. The HTTP header is highly dynamic and will differ depending on which direction and which message is sent in addition to varying between different browsers and servers
- 225 bytes XMPP XML/content. The content of the XML based XMPP header is highly dynamic.

This leaves some room for data; depending on the variation in the application headers, approximately:

- 685 bytes data - estimated with dynamic XMPP and HTTP headers

The three first header fields, the Ethernet, IP and TCP header are all fixed-size headers. The two last however, are dynamic in length depending on which type of message is relayed and should only be considered a sample number, experienced when looking at arbitrary packets in an XMPP/HTTP stream. A HTTP header from the browser to the server might have a size of as much as 550 bytes, containing information about the user's system and browser. This is as much as one third of the maximum IPv4 payload of 1460 bytes. In addition to the HTTP header comes the XMPP header which might be 225 bytes for a given packet. This shows how inefficient the use of the HTTP protocol is. At the same time, it is not easily circumvented when relying on a browser because of the limitations of the browser environment. Figure 59 shows a comparison of the protocol header between the Home server and Network server on the left, and between the Web client and the Network server on the right

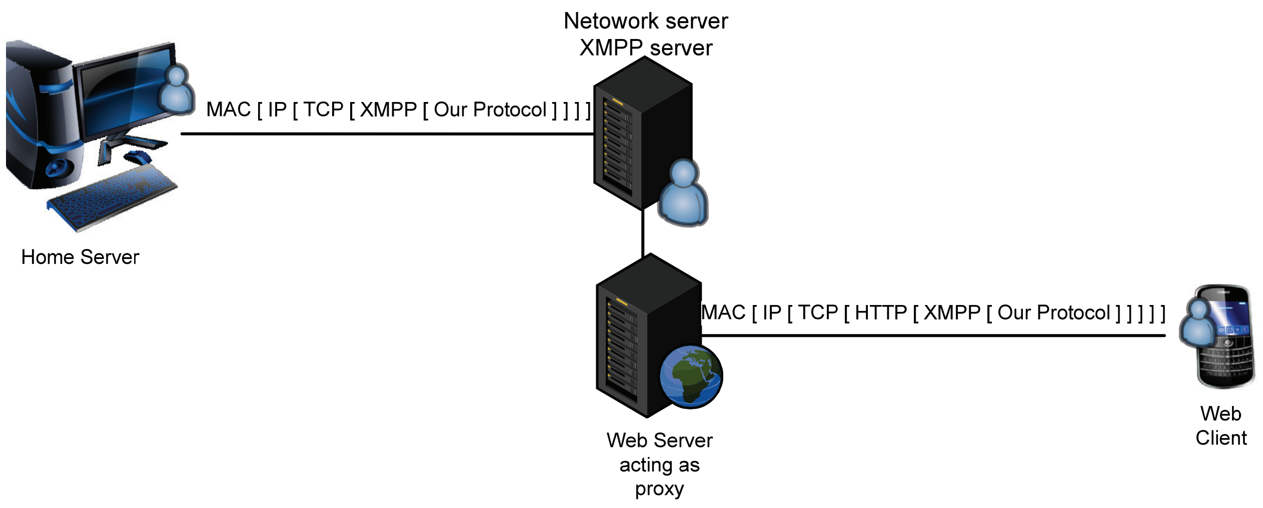

Netowork server
XMPP server
MAC [IP [TCP [XMPP [Our Protocol]]]]
Home Server
MAC [IP [TCP [HTTP [XMPP [Our Protocol]]]]]
Web Server acting as proxy
Web Client

Figure 59 - XMPP client communicating with a web/HTTP based XMPP client

In other words, the controversy is still present; the link with the least bandwidth needs to use the protocol with the most overhead to incorporate the use of a web browser. The drawbacks of this have to be valued against the benefits of programming for the web browser. The bottom line is, using a web browser as a client interface in an XMPP network utilizing BOSH adds a large overhead of somewhere between 300-500 bytes per packet sent. Still, it is considered beneficial for this thesis. For a benchmark estimate of using BOSH in a large community of users, visit the ESL Developer blog at [144].

XMPP is an evolving technology and while XMPP over BOSH might seem to have a large overhead. Some argue that the XMPP protocol itself is also too inefficient. An effort is under way to create an extension to the XMPP standard, optimized for mobile use. This has particularly been discussed in the jabber mailing lists after it was known that the new Google Android SDK would not include a pure XMPP library, but would rather use a proprietary solution to communicate with the Google talk servers because of the large overhead [145].

3.2.5.1 The core communication protocol

To be able to manipulate the core part of the application, a protocol has been designed for the purpose, it follows the outline in the protocol design chapter where packets are divided into two, one for the core application, with a <MessageType> Communication and the second with <MessageType> service for all the available services. To differentiate which packet is for which service, the search service uses a <Type> tag with contents Search. It is up to the individual service to filter out the relevant packets for its use.

Get Roster and Receive Roster

The Roster of the home server can be retrieved by sending the following packet to the home server user.

```
<?xml version="1.0"?>
<Body>
	<MessageType>Communication</MessageType>
	<Type>GetRoster</Type>
</Body>
```

The Home server answers with the following packet

```
<?xml version="1.0"?>
<Body>
	<MessageType>Communication</MessageType>
	<Type>Roster</Type>
	<item jid="homeserver@129.241.208.209" name="homeserver" subscription="both">
		<group>HS</group>
	</item>
	<item jid="larsare@129.241.208.209" name="larsare" subscription="both">
		<group>WEB</group>
	</item>
</Body>
```

Get services and Available services

The GetServices packet is sent from a client who is interested in knowing which services the server can provide

```
<?xml version="1.0"?>
<Body>
	<MessageType>Communication</MessageType>
	<Type>GetServices</Type>
</Body>
```

The Home server answers with the following packet, giving away all the services which have been loaded into the plug-in framework

```
<?xml version="1.0"?>
<Body>
	<MessageType>Communication</MessageType>
	<Type>Services</Type>
	<Payload>
		<Service id ="0">
			<Name>Search</Name>
		</Service>
		<Service id ="1">
			<Name>PVR</Name>
		</Service>
		<Service id ="n">
			<Name>X</Name>
		</Service>
	</Payload>
</Body>
```

3.2.5.2 *The Search service protocol*

The following will describe the protocol made for the example service, which was a network based search. Important packets here are the search query and search result, as well as the file Transfer request.

Search Request

The search request packet contains important information concerning the relevance of the query as well as the query itself. As of the writing of this report, the FileType and Hits fields do not affect the search, as these parameters are coded into the search plug-in. They are there to easily enable a future expansion of the search service.

```
<?xml version="1.0"?>
<Body>
        <MessageType>Search</MessageType>
        <Type>Request</Type>
        <Payload>
                <Query>Query goes here</Query>
                <FileType>specify filetypes to search in</FileType>
                <Hits>number of hits to get back</Hits>
        </Payload>
<Body>
```

Next, the Search Response packet

```
<?xml version="1.0"?>
<Body>
        <MessageType>Search</MessageType>
        <Type>Response</Type>
        <Payload>
                <String id="0">
                        <FileName></FileName>
                        <Path></Path>
                        <Size></Size>
                        <Fragments><![CDATA[Highlighted Fragment]]></Fragments>
                        <modified>200905031439 date and time</modified>
                </String>
                  .
                  .
                  .
                <String id="n">
                  .
                  .
                </String>
        </Payload>
</Body>
```

FileTransfer

The file transfer involves three entities in the network. The Home server, the network server client and the remote client and is considered part of the search service in this thesis. The Remote client has performed a search and has gotten a few hits back. Together with the hits is the path to the file. The file transfer is initiated when the remote client requests this file from the home server. The file transfer might also be considered a separate service if restructured in a later edition of the software.

The GetFile, file transfer request is sent from the client to the home server.

```
<?xml version="1.0"?>
<Body>
        <MessageType>Search</MessageType>
        <Type>GetFile</Type>
        <Payload>
                <Path>D:\MRL\temp\test4.txt</Path>
        </Payload>
</Body>
```

The home server receives the GetFile packet, A file transfer is then initiated towards the Network Server client, aka, the “NSclient” The file transfer is made possible through the XMPP reliable file transfer extension. The NSclient’s main responsibility is to listen for file transfer requests from its associated users and store received files in folders tagged with the name of the home server from whom they were received. If the file transfer is completed, the NSclient sends a packet containing the list of files available in the current home server’s folder.

```
<?xml version="1.0"?>
<Body>
        <MessageType>Search</MessageType>
        <Type>*UserFiles</Type>
        <Payload>
                <String id="0">
                        <FileName></FileName>
                </String>
                   .
                   .
                   .
                <String id="n">
                        <FileName></FileName>
                </String>
        </Payload>
</Body>
```

On the other hand, if the file transfer is not successful, the home server sends a file transfer failure message to the client.

```
<?xml version="1.0"?>
<Body>
        <MessageType>Search</MessageType>
        <Type>Failure</Type>
</Body>
```

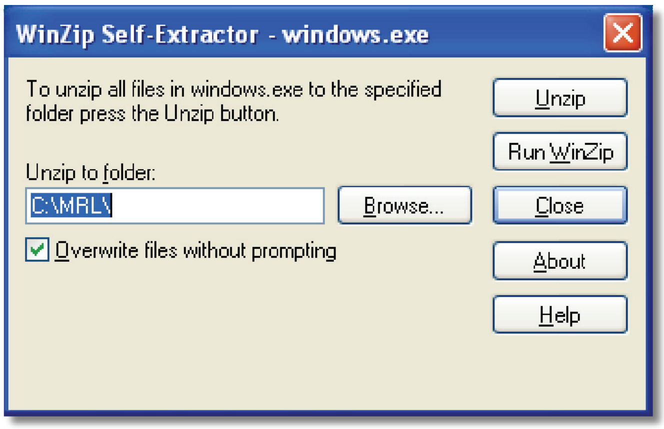
3.2.6 System bootstrap

This is a description of the procedure necessary to bootstrap the system from a user point of view.

3.2.6.1 How to bootstrap the system from a users point of view

To setup the HS, a user must download and install a desktop application available on the web portal. After the application is downloaded, the files must be extracted to reveal the executable file: *HomeApplicationStart*. If the user has Windows, the extraction of the files is performed automatically using WinZip self extractor, shown in Figure 60. To be able to run the *HomeApplicationStart,* Java 1.5

or 1.6 must be installed on the computer, and to connect to the NS the computer must have Internet access.


WinZip Self-Extractor - windows.exe

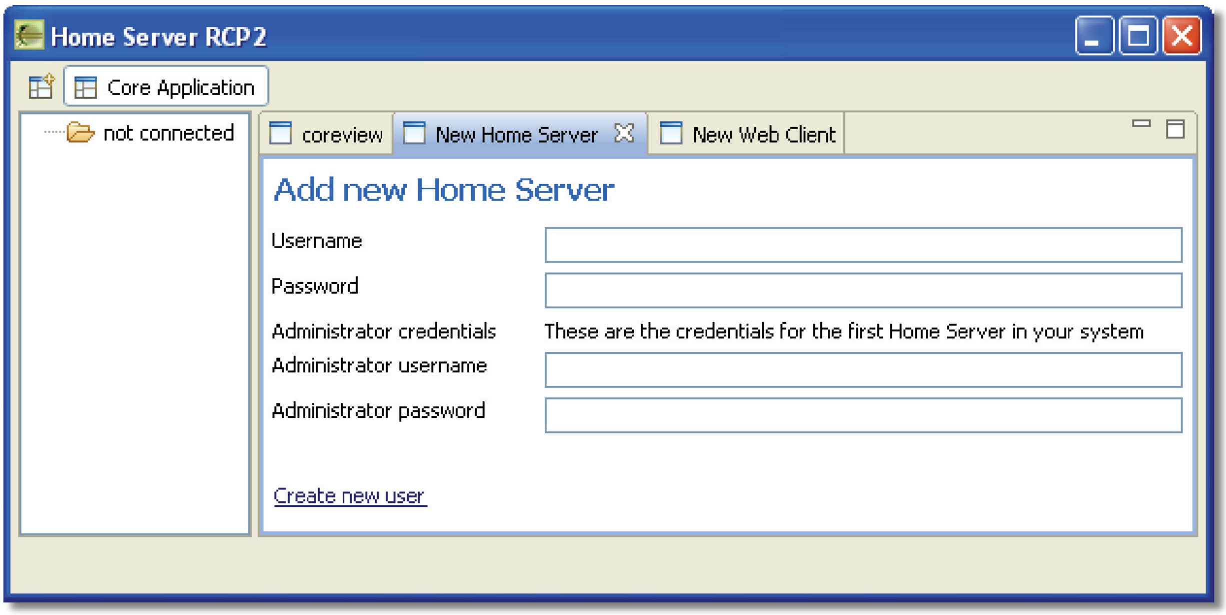
To unzip all files in windows.exe to the specified folder press the Unzip button.

Unzip to folder: C:\MRL\

✓ Overwrite files without prompting

Figure 60 - Self extraction of the desktop application

When the application is launched for the first time, a window with different views is presented to the user. If it is the first time the user try to connect to the system, a user must register an account to get access. The user goes to the “New Home Server” view, type in a username and password, and hit the button “Create new user”. In Figure 61, a screen shot of the “New Home Server” view is shown.


Home Server RCP2

Add new Home Server

Username

Password

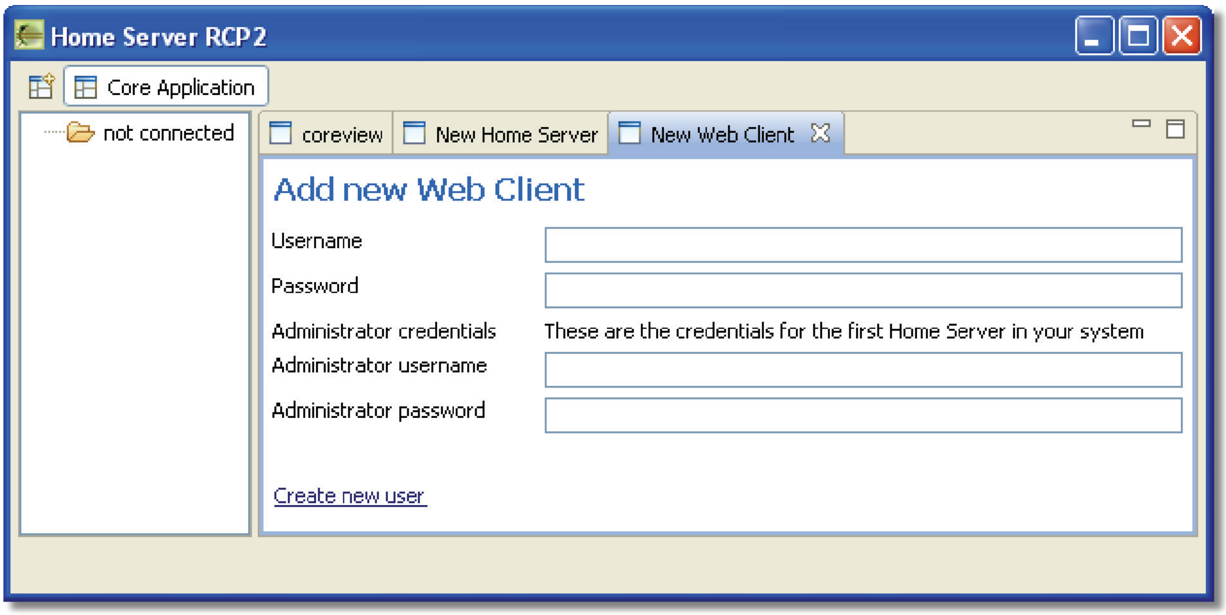
Administrator credentials These are the credentials for the first Home Server in your system

Administrator username

Administrator password

Figure 61 - Screen shoot of "New Home Server" view

The HS is now registered to the NS, and the user can login with the newly created account. The HS is now connected to the NS, but the system is not usable before at least one web client account is created and one service is downloaded from the system home page. The creation of a web client is performed in almost the same way as the HS account. The user goes to the “New Web Client” view, type in a username and password and the admin credentials, which are the same username and password as to the first HS, then hit the button “Create new user”. In Figure 62, a screenshot of the “New Web Client” view is shown.


Home Server RCP2

Core Application

not connected

coreview | New Home Server | New Web Client

Add new Web Client

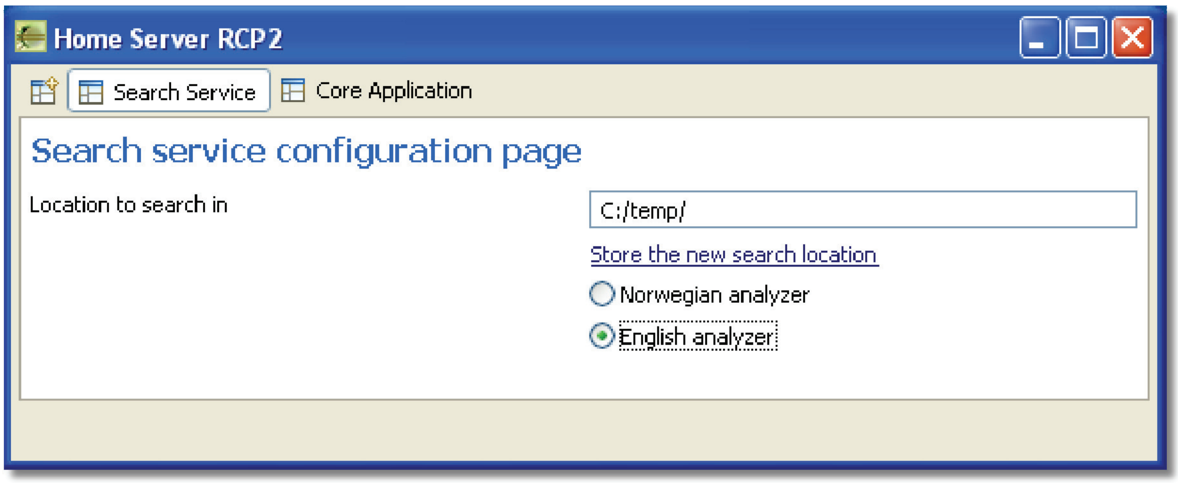
Username	
Password	
Administrator credentials	These are the credentials for the first Home Server in your system
Administrator username	
Administrator password	

Create new user

Figure 62 - Screen shoot of "New Web Client" view

The user can also later on create users, both web clients and HS clients, but then the admin password and username is required.

The HS has now registered a new web client, and the web client can login with the newly created account from a browser. If the user wants to add services to the HS, the user must download a package containing the service from the web portal. The package is extracted to the “plugins” folder in the HS directory. After restarting the HS, the service should be available to the HS and all web clients in the HS roster. To view the new service perspective, click on the “open perspective” button in the upper left corner and select a service. In Figure 63, a screen shot of the search service perspective is shown.


Home Server RCP2

Search Service | Core Application

Search service configuration page

Location to search in: C:/temp/

Store the new search location

Norwegian analyzer [✓ English analyzer]

Figure 63 - Screen shoot of the search service perspective

As shown in Figure 63, this perspective shows the service, in this example the search service, and provides a window to the user of available configurations related to the service.

How to get the system up and running, a step-by-step summary:

1. Go to system home page and download the desktop application
2. Unpack the file into a folder of your own choice
3. Start the application by starting the executable file: *HomeApplicationStart*
4. Register a new HS and a new web client
5. Go to system home page and download a service
6. Login to the HS with the newly created credentials
7. Login to a web client, located on the system home page

The system should now be up and running, and the HS and the downloaded services should be available to all created web clients.

Mobile Remote LAN Master Thesis

Part IV – Evaluation

The testing, conclusion, and future work are revealed in part four. It explains the accomplishments of the thesis, and focus on future development to improve the service platform. Evaluation of cost is also revealed in this part in order to show the behavior of the service platform in a real life environment. Cost examples such as bandwidth and time consumptions, cost in NOK, and resource consumptions are shown. In addition a short reflection note is added to highlight the authors thought about this thesis.

4 Part IV - Evaluation

4.1 Testing and cost

This chapter will look into some of the aspects of this platform when it comes to actual cost of usage and performance when running the search and indexing on a computer.

4.1.1 Estimated cost

To estimate the cost of the service, Mozilla Firefox is used to download the web portal from the NS and Wireshark is used to capture all packets between the web client and the NS. An estimated cost based on the packets sent between the web client and the NS received using Wireshark is shown in Table 3. The numbers are approximately due to the differences in the TCP protocol, but the differences are so small that they can be neglected.

Table 3 - Estimated cost of downloading the web portal, log in, and quit.

Action	Total Bytes [kilo Bytes]
Download web portal	129,5
Log in	8,5
Connect to a HS	3,0
Quit	2,0
Total cost	143,0

The total cost of log in and access a HS, look at the available services and then log out is approximately 143 bytes. There are many variables to consider if an exact cost of using the system should be achieved, but most of the variables are small enough to be neglected. I.e. the length of the JID can have at maximum 3071 bytes in total [85], but the user only have access to change 1023 bytes which indicate the username. Other examples are the size of the roster and available services, but in this context these sizes are small enough to neglect, as is the variable JID.

These tests are preformed on a computer with 100Mbps connection, so the time usage of downloading and interacting with the web portal is very small, approximately a few ms. On slower connections though, the time can be considerably higher.

4.1.2 Cost, bandwidth

To estimate the time of various actions regarding the web portal, the total bytes in Table 3 and throughput rates in different cellular connections are used. The data in Table 4 is based on the article [146] and shows the peak and average throughput in the various cellular systems.

Table 4 - Data rate comparison in UMTS and GSM networks

Technology	Peak, download speed [kbps]	Average throughput, download [kbps]	Average throughout, download [kilo Bytes/sec]
GPRS	115	30 - 40	3,75 - 5
EDGE	473	100 - 130	12,5 - 16,3
UMTS-WCDMA[1]	2000	220 - 320	27,5 - 40
UMTS-HSDPA[2]	14000	500 - 1100	62,5 - 137,5

[1] 384 kbps typical maximum rate of current devices

[2] Current devices are likely to have a maximum rate of 1.8 or 3.6 Mbps

The throughput in Table 4 depends on many factors such as users, distance, interference, etc., but the numbers give a good indication of the differences between the cellular connection rates. To estimate time consumed when a user performs various actions, the average throughput rate can be used since the most likely bottleneck, when a remote device is connected to the system through the cellular network, will be the cellular network.

The average throughput from Table 4 and the total bytes from Table 3 are used to estimate the total amount of time various actions towards the web portal consumes. The consumed times are shown in Table 5, and the equation is: $\frac{Total\ bytes\ from\ Table\ 3}{Average\ throughput\ from\ Table\ 4} = time\ consumed\ in\ Table\ 5$

Table 5 - Time consumption of various actions performed

Action	GPRS [sec]	EDGE [sec]	WCDMA [sec]	HSDPA [sec]
Download web portal	34,5 - 25,9	10,4 - 7,9	4,7 - 3,2	2,1 - 0,9
Log in	2,3 - 1,7	0,7 - 0,5	0,3 - 0,2	0,1 - <0,1
Connect to a HS	0,8 - 0,6	0,2 - 0,2	<0,1 - <0,1	<0,1 - <0,1
Quit	0,5 - 0,4	0,2 - 0,1	<0,1 - <0,1	<0,1 - <0,1
Total time consumed	38,1 - 28,6	11,4 - 8,8	5,2 - 3,6	2,3 - 1,0

There are big differences between the various connections, and the total time consumed differs with more than 37 seconds at most. The main reason for this is that the more or less outdated technology GPRS is considered in Table 5. In Norway, and many other western countries, GPRS is replaced with EDGE [147] [148], and the difference in time consumed decreases with more than two hundred percent to about 11,3 seconds at most. The user can expect, depending on the technology, topography, distance to transmitter, buildings, and remote device that the total time consumed, executing the various actions described in Table 5, is less than 10 seconds.

The time consumed when a user interacts with the system is very important to the user experience, but the users also care about the cost. In the next section an estimate of the cost is presented.

4.1.3 Cost, NOK

Many users think of cost as an important aspect when deciding to buy a new device or try a new technology or service. The cost estimated in this section is based on subscriptions with a fixed cost per megabyte transferred, and no other costs. To give an indication of the cost, three of the top subscriptions at http://www.telepriser.no with very different data transfer cost are selected and shown in Table 6.

Table 6 - Cost of data transfer from three of the cheapest mobile phone subscriptions

	One Call – Faktura	Djuice – SIMply	Tele2 – Sheriff
Cost, MB/sec [in NOK]*	0,99	5	12

*The data is collected from: http://www.telepriser.no

To give a cost estimation of the various actions, data from Table 3 is used. The different costs are shown in Table 7, and the equation is: $\mathit{The\ cost\ from\ Table\ 6}\ \frac{\mathit{Total\ bytes\ from\ Table\ 3}}{1\ \mathit{Mega\ byte}} = \mathit{cost}$

Table 7 - User cost of various actions performed

Action	One Call – faktura [NOK]	Djuice – SIMply [NOK]	Tele2 – Sheriff [NOK]
Download web portal	0,13	0,65	1,55
Log in	<0,01	0,04	0,10
Connect to a HS	<0,01	0,02	0,04
Quit	<0,01	0,01	0,02
Total cost	0,14	0,72	1,72

There are big differences, Djuice and Tele2 are more than 5 and 12 times more expensive to use than the One Call subscription. The previous sections have revealed that there are big differences in both time consumption and cost, depending on respectively the connection and subscription type. The previous sections have only estimated cost of basic interactions with the web portal, and have not specified any costs of using a service deployed in the system. The next chapter estimates the additional user cost and time consumption when using the search service.

4.1.4 Cost, idle

The NS send a HTTP-bind message, keep-alive message, to the web client every 280 seconds to ensure the connection stays up. This has a cost for the user, even though it is quite small. The cost every 280 second is approximately: 1161 + 60 (ACK) bytes = 1221, giving 4.36 bytes a second. The table below shows the cost of staying connected to the NS.

Table 8 - Cost in kilobytes of keep-alive messages between the NS and web client when idle

10 min [kilobytes]	30 min [kilobytes]	1 hour [kilobytes]	5 hours [kilobytes]	1 day [kilobytes]
2,6	7,8	15,7	78,5	376,7

In Table 3 the total cost of basic interactions was estimated to 143 kilobytes, and comparing this with Table 8, it shows that if the system is used more than once every 9th hour, the cost of download the web portal is bigger than to stay connected. If the web portal is cached in the browser, the cost is only approximately 13.5 kilobytes. The system must be used more than every hour if the user should chose to not log in and out of the system.

4.1.5 Cost, search service

The cost of performing a search and downloading a file using the search service depends on several variables, e.g.:

- Length of the query string, can be from one byte and up
- The number of hits returned, there can be 10 hits pr available HS
- File descriptions
  - Name, no limit
  - Path, no limit
  - Size, no limit
  - Score, 9 Bytes
  - Modified, 12 Bytes
  - Fragments, can be maximum of 306 Bytes in payload length and is where the summary of the file is stored
- Size of the file to download

All the variables are dependent on the query string, the indexed files, and the availability of HSs. The amount of information for each file is decisive to the size of the query response. I.e. the fragment variable can be from 0 to 306 bytes, and can in theory raise the cost of 300 bytes per hit, which can be a considerable part of the total data amount in the query response. In practice the difference in actual bytes are smaller due to the compression, using gzip [149], of the entity body.

The minimum amount of bytes for a:

- Query is 1151 bytes + one byte pr letter in the query string, not encoded
- Search response is 557 bytes, encoded

In addition the HTTP/XML protocol transfers:

- 200 ok protocol packet from the NS to the web client of 400 bytes, encoded
- http-bind protocol packet from the web client to the NS of 784 bytes, not encoded

The numbers above are approximate, they depend on the JID and other variables described earlier in this chapter, but the relationship between the numbers are fixed.

In Table 9 the total bytes transferred using the HTTP/XML protocol is summed up, the query used to get non hits consists of three bytes. ACK and SYN packets are not considered in the following examples due to the differences in number and size, and the ratio of size compared to the HTTP/XML packets.

Table 9 - Total size of a search response from one HS with no hit

Number of hits [pieces]	Packets [pieces]	Total Bytes [kilo Bytes]	Avg. packet size [Bytes]
0	4	2,895	723,750

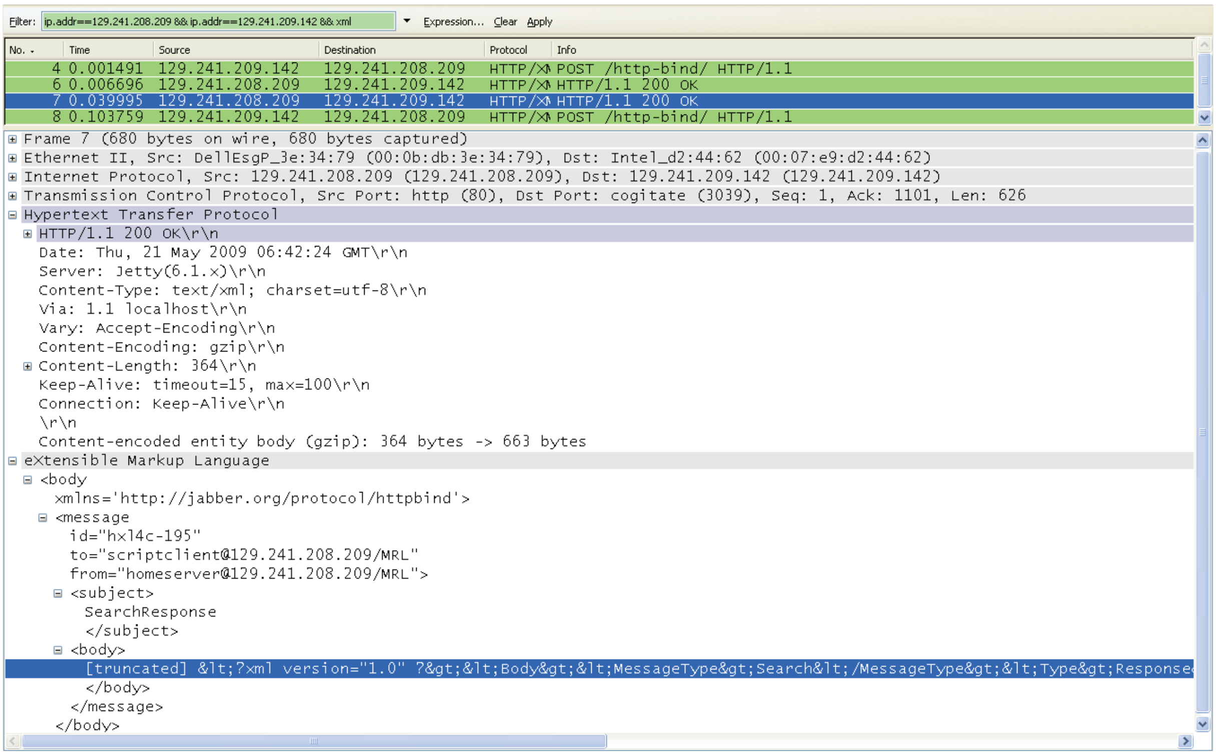
To give a better example of the cost of using the search service, an estimate of how much the fragments variables differ is shown. The search service example just gives an indication of the cost, and is not an accurate cost calculation. Table 10 shows the difference in bytes between a hit with no fragments, and a hit with the maximum fragment size.

Table 10 - Difference in bytes between zero and maximum fragment size

	Fragment size [Bytes]	Number of hits [pieces]	Packets [pieces]	Total Bytes [kiloBytes]	Avg. packet size [Bytes]
	306	1	4	3,188	797,0
	0	1	4	3,018	754,5
Difference	306	0	0	0,170	42,5

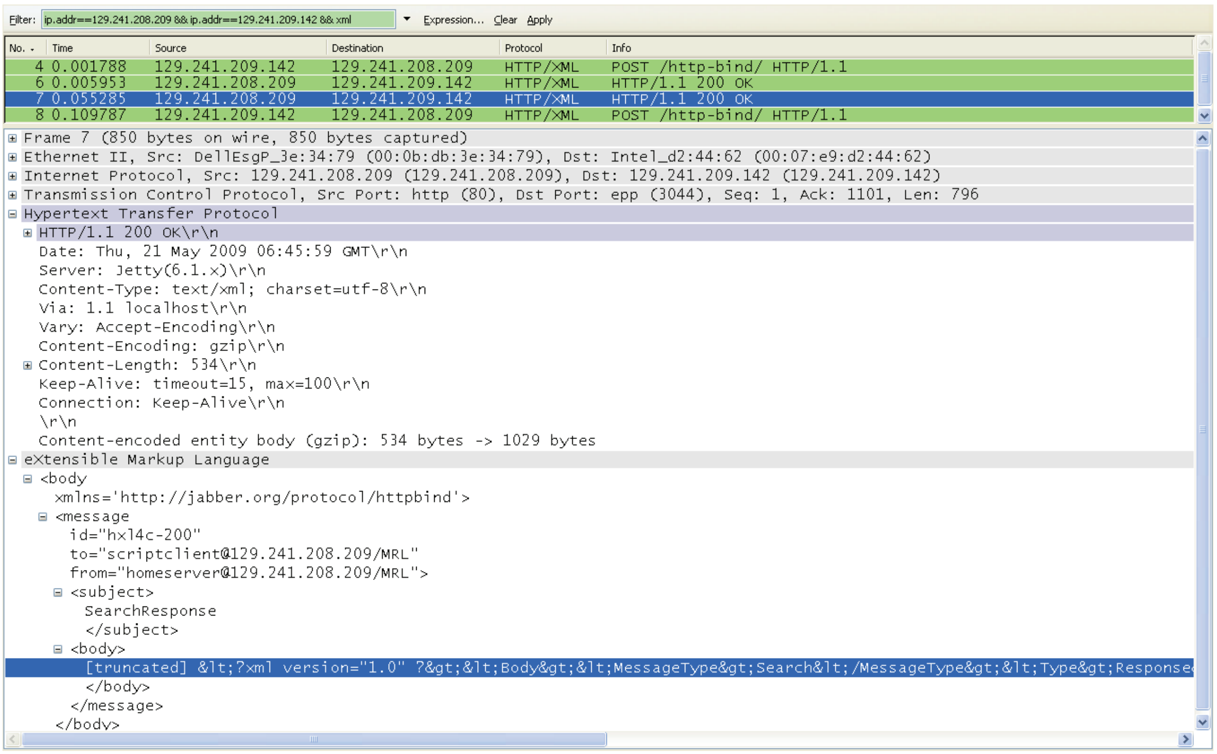
In Table 10 the difference in total bytes are only 170 bytes, and 306 bytes due to the encoding. If we take a closer look of the packets sent between the web client and the NS in Wireshark, it reveals that only the search response packet is different in size. The difference is 680 to 850, a raise of 25%, only due to the fragments size. This is important to have in mind when the cost example is presented next; the cost of the search service is dependent of many variables and it is difficult to anticipate how big the search response is to the user. In addition, the variable with the biggest impact on cost is the file size of the file the user wants to download.

The next four figures reveal the packet capture in Wireshark for the previous example of variable fragments size. The lines marked with blue are where the packets differ in size.


Filter: ip.addr==129.241.208.209 && ip.addr==129.241.209.142 && xml
Expression... Clear Apply

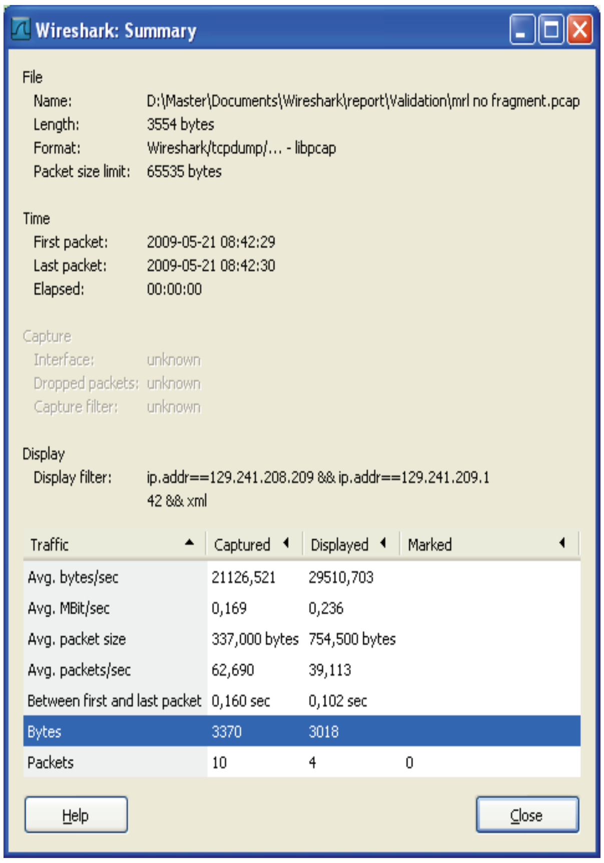
```
No.  Time      Source           Destination      Protocol  Info
  4 0.001491 129.241.209.142  129.241.208.209  HTTP/X  POST /http-bind/ HTTP/1.1
  6 0.006696 129.241.208.209  129.241.209.142  HTTP/X  HTTP/1.1 200 OK
  7 0.039995 129.241.208.209  129.241.209.142  HTTP/X  HTTP/1.1 200 OK
  8 0.103759 129.241.209.142  129.241.208.209  HTTP/X  POST /http-bind/ HTTP/1.1

⊞ Frame 7 (680 bytes on wire, 680 bytes captured)
⊞ Ethernet II, Src: DellEsgP_3e:34:79 (00:0b:db:3e:34:79), Dst: Intel_d2:44:62 (00:07:e9:d2:44:62)
⊞ Internet Protocol, Src: 129.241.208.209 (129.241.208.209), Dst: 129.241.209.142 (129.241.209.142)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: cogitate (3039), Seq: 1, Ack: 1101, Len: 626
⊟ Hypertext Transfer Protocol
  ⊞ HTTP/1.1 200 OK\r\n
    Date: Thu, 21 May 2009 06:42:24 GMT\r\n
    Server: Jetty(6.1.x)\r\n
    Content-Type: text/xml; charset=utf-8\r\n
    Via: 1.1 localhost\r\n
    Vary: Accept-Encoding\r\n
    Content-Encoding: gzip\r\n
  ⊞ Content-Length: 364\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    \r\n
    Content-encoded entity body (gzip): 364 bytes -> 663 bytes
⊟ eXtensible Markup Language
  ⊟ <body
      xmlns='http://jabber.org/protocol/httpbind'>
    ⊟ <message
        id="hx14c-195"
        to="scriptclient@129.241.208.209/MRL"
        from="homeserver@129.241.208.209/MRL">
      ⊟ <subject>
          SearchResponse
          </subject>
      ⊟ <body>
          [truncated] &lt;?xml version="1.0" ?&gt;&lt;Body&gt;&lt;MessageType&gt;Search&lt;/MessageType&gt;&lt;Type&gt;Response
          </body>
        </message>
      </body>
```

Figure 64 - Capture of SearchResponse message with no summary in the fragments variable



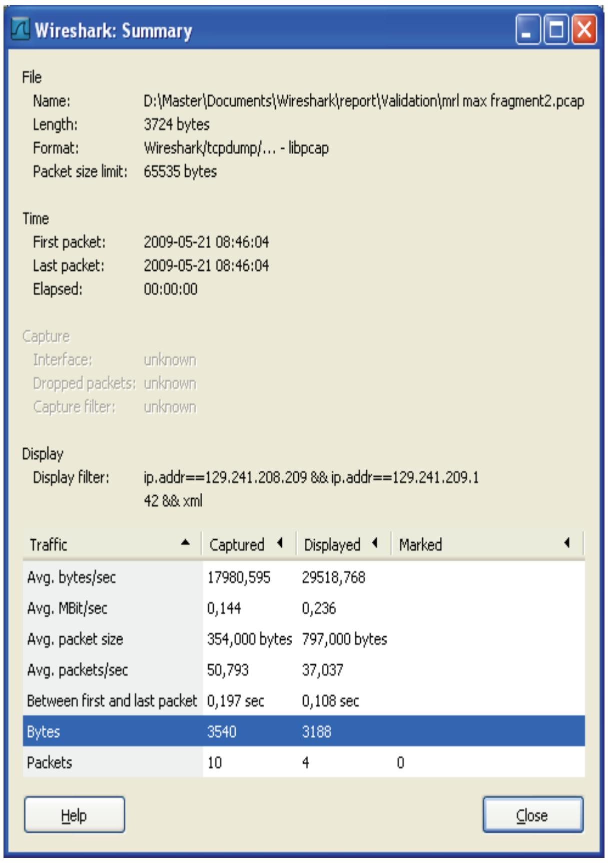
```
Filter: ip.addr==129.241.208.209 && ip.addr==129.241.209.142 && xml   Expression...  Clear  Apply

No. -  Time        Source            Destination       Protocol   Info
   4 0.001788   129.241.209.142   129.241.208.209   HTTP/XML   POST /http-bind/ HTTP/1.1
   6 0.005953   129.241.208.209   129.241.209.142   HTTP/XML   HTTP/1.1 200 OK
   7 0.055285   129.241.208.209   129.241.209.142   HTTP/XML   HTTP/1.1 200 OK
   8 0.109787   129.241.209.142   129.241.208.209   HTTP/XML   POST /http-bind/ HTTP/1.1

⊞ Frame 7 (850 bytes on wire, 850 bytes captured)
⊞ Ethernet II, Src: DellEsgP_3e:34:79 (00:0b:db:3e:34:79), Dst: Intel_d2:44:62 (00:07:e9:d2:44:62)
⊞ Internet Protocol, Src: 129.241.208.209 (129.241.208.209), Dst: 129.241.209.142 (129.241.209.142)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: epp (3044), Seq: 1, Ack: 1101, Len: 796
⊟ Hypertext Transfer Protocol
  ⊞ HTTP/1.1 200 OK\r\n
    Date: Thu, 21 May 2009 06:45:59 GMT\r\n
    Server: Jetty(6.1.x)\r\n
    Content-Type: text/xml; charset=utf-8\r\n
    Via: 1.1 localhost\r\n
    Vary: Accept-Encoding\r\n
    Content-Encoding: gzip\r\n
  ⊞ Content-Length: 534\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    \r\n
    Content-encoded entity body (gzip): 534 bytes -> 1029 bytes
⊟ eXtensible Markup Language
  ⊟ <body
      xmlns='http://jabber.org/protocol/httpbind'>
    ⊟ <message
        id="hx14c-200"
        to="scriptclient@129.241.208.209/MRL"
        from="homeserver@129.241.208.209/MRL">
      ⊟ <subject>
          SearchResponse
          </subject>
      ⊟ <body>
          [truncated] &lt;?xml version="1.0" ?&gt;&lt;Body&gt;&lt;MessageType&gt;Search&lt;/MessageType&gt;&lt;Type&gt;Response
          </body>
        </message>
      </body>
```

Figure 65 - Capture of Search Response message with maximum size of the fragments variable



```
Wireshark: Summary

File
  Name:               D:\Master\Documents\Wireshark\report\Validation\mrl no fragment.pcap
  Length:             3554 bytes
  Format:             Wireshark/tcpdump/... - libpcap
  Packet size limit:  65535 bytes

Time
  First packet:       2009-05-21 08:42:29
  Last packet:        2009-05-21 08:42:30
  Elapsed:            00:00:00

Capture
  Interface:          unknown
  Dropped packets:    unknown
  Capture filter:     unknown

Display
  Display filter:     ip.addr==129.241.208.209 && ip.addr==129.241.209.1
                      42 && xml

Traffic                      Captured       Displayed      Marked
Avg. bytes/sec               21126,521      29510,703
Avg. MBit/sec                0,169          0,236
Avg. packet size             337,000 bytes  754,500 bytes
Avg. packets/sec             62,690         39,113
Between first and last packet 0,160 sec     0,102 sec
Bytes                        3370           3018
Packets                      10             4              0

Help                                                       Close
```

Figure 67 - The summary of fragments size=0



```
Wireshark: Summary

File
  Name:               D:\Master\Documents\Wireshark\report\Validation\mrl max fragment2.pcap
  Length:             3724 bytes
  Format:             Wireshark/tcpdump/... - libpcap
  Packet size limit:  65535 bytes

Time
  First packet:       2009-05-21 08:46:04
  Last packet:        2009-05-21 08:46:04
  Elapsed:            00:00:00

Capture
  Interface:          unknown
  Dropped packets:    unknown
  Capture filter:     unknown

Display
  Display filter:     ip.addr==129.241.208.209 && ip.addr==129.241.209.1
                      42 && xml

Traffic                      Captured       Displayed      Marked
Avg. bytes/sec               17980,595      29518,768
Avg. MBit/sec                0,144          0,236
Avg. packet size             354,000 bytes  797,000 bytes
Avg. packets/sec             50,793         37,037
Between first and last packet 0,197 sec     0,108 sec
Bytes                        3540           3188
Packets                      10             4              0

Help                                                       Close
```

Figure 66 - The summary of fragments size=306

4.1.5.1 Example, search service

To estimate the cost of using the search service, the best way is to perform an example and show the total cost of using the service. An estimate of the cost of downloading the page, logging in, connecting to HS, and logging out is already performed. This example estimates the cost of:

- Send a query to NS
- Get a search service response packet
- Uploading a file to NS from HS

The costs of downloading the file is not taken into account, the file size to download comes on top of this estimation. The cost of downloading the file is approximately the same as the file size, only some TCP ACK and SYN packets are sent in addition.

In the following example the query is: "capital", and results in 10 hits from one HS, and all hits have maximum fragments size. In Table 11 the total costs of using the service is shown, the basic actions cost is gathered from Table 3.

Table 11 - Total cost, in kilo bytes, of using the search service

Action	Total Bytes [kilo Bytes]
Basic actions	143,0
Search service	8,1
Total cost	151,1

Table 11 shows that the search service actions have little impact on the total cost of using the search service. This is also shown in bandwidth and NOK consumptions in Table 12 and Table 13.

Table 12 - Total cost, in time consumption, of using the search service

Action	GPRS [sec]	EDGE [sec]	WCDMA [sec]	HSDPA [sec]
Basic actions	38,1 - 28,6	11,4 - 8,8	5,2 - 3,6	2,3 - 1,0
Search service	2,2 - 1,6	0,6 - 0,5	0,3 - 0,2	0,1 - 0,1
Total time consumed	40,3 - 30,2	21,0 - 9,3	5,5 - 3,8	2,4 - 1,1

Table 13 - Total cost, in NOK, of using the search service

Action	One Call – faktura [NOK]	Djuice – SIMply [NOK]	Tele2 – Sheriff [NOK]
Basic actions	0,14	0,72	1,72
Search service	0,01	0,04	0,10
Total cost	0,15	0,76	1,82

As a conclusion to the previous example, the cost using the search service is mainly dependent on the basic costs of the system, such as downloading the page, login and out, and connection to a HS.

4.1.6 Indexing and search performance

The Lucene search engine utilizes an indexer to speed up the search process. The indexer is used to parse all the documents into an index. It takes the text content out of the different file types and stores it in the index for search purposes. This index is used when searching. This way the original documents are not involved in the search itself. The size of the index is comparable to how much text is actually inside a document. Html documents for example, which contain mostly text, will generate an index quite large compared to the size of the files, while a Microsoft docx document, like this thesis which consist of some images in addition to the text will generate a smaller footprint in the index.

The major benefit of using an indexer is the search speed. The indexer runs through all the files only once. When the index is made, it is only optimized on each search. The optimization process compares the UID "Unique Identifier" of the indexed files with the files in the file system and skips every file that is not altered. Table 14 shows some random numbers from different file types and sizes. The table should not be used for statistical comparison, but only to get an idea of the variety of speed and sizes involved.

Table 14 - Some various measurements of document indexing and search

Folder size	Index size	Time to index	Time to optimize index	Time to search	Folder contents
62MB	15MB	32 s	172 ms	187 ms	All kinds of documents
62MB	2,6MB	30 s	157 ms	93 ms	docx files comparable to this report
62MB	32MB	22 s	188 ms	281 ms	.html files only
62MB	5,5 MB	73 s	172 ms	188 ms	.pdf files only

The time taken to index the files depend on what kind of files, how big and how many. Especially some of the newer file types, which are only supported in beta by the Tika library, can take some time to index. It is important to note that the size of the index will not grow linearly with the number of documents, as phrases, or words are only stored once and pointers to the placement of them is stored for each file. This means an index containing many similar documents will take up less space than an index of all different files. When doing search optimization in a future edition of this application it is possible to optimize the size of the index by selecting more or fewer document fields to index. The current index is set up to be able to provide a highlighted text fragment of the search query. This means the whole text content of each file must be stored in the index. This will of course influence the size of the index. Table 15 is a comparison of a collection of documents stored first as Microsoft Office docx files, and then the same files has been converted into Portable Document Format (PDF). As expected, the generated index is approximately the same size, as the text contents of the files are the same size. The indexing and search also take close to the same time. The somewhat difference in time, is too small to be significant with the few files that have been indexed here. Observations are measured by the Date function in Java, which means the accuracy of the measurements is not the best. The all time lowest seen in accessing the index was 125 ms, which probably means much of this time is spent on accessing the file system, etc.

Table 15 - comparing some of the parts of this report in docx and pdf

Folder size	Index size	Time to index	Time to optimize index	Time to search	Folder contents
11MB	508 KB	6,9 s	125 ms	109 ms	DOCX documents
5,8MB	607 KB	5 s	140 ms	141 ms	PDF documents

The Tika parser is capable of indexing the metadata of many different file types. The next example, shown in Table 16 is an index consisting of mp3 files. These files contain some metadata, e.g. actor, song title, length and so forth. This metadata is often used by mp3 players to categorize songs. As shown in the table, the size of the index is quite small compared to the text files indexed above.

Table 16 - Indexing and searching in Mp3 files

Folder size	Index size	Time to index	Time to optimize index	Time to search	Folder contents
239 MB	23 KB	1,7 s	141 ms	94 ms	1 album of Mp3's
366 MB	32 KB	2,1 s	125 ms	78 ms	2 albums of Mp3's
3,16 GB	386 KB	124 s	250 ms	47 ms	538 different Mp3's

4.2 Reflections

Creating a generic framework that works in all network settings is quite a challenge, and most of the hours spent on this thesis have consisted of coding and researching to get the prototype up and running. In other words this report is only one side of the thesis, an important one; however the development of the prototype has by far been the most time consuming event of this thesis.

The authors' knowledge of technologies such as Eclipse RCP, JavaScript, and XMPP was limited, and the learning curve was very steep. Many hours have been spent getting to know the technologies and study documentation/RFCs to acquire enough knowledge to make the prototype.

In addition to the time consumed in research and development of the prototype and writing the report, some time has been used to make a presentation and demo for Telenor. It was rewarding to get the acknowledgement of the people giving the assignment in the first place, and to be told that we had done a good job.

This thesis has been very interesting and has invoked a special interest in the field of mobility and web based services. It has been five very exciting months, and the choice of writing a thesis together has been a very good experience. The possibilities of learning, backing each other up, and looking at problems with different eyes have been a key element in the development of the prototype and the report.

4.3 Future work

During the work on this Master Thesis, many solutions and design choices have been discussed. The solutions necessary to create and show the functionality of the service platform has been implemented. Others were left for future work. The emphasis throughout the thesis has been on providing the functionality needed to make a working prototype.

4.3.1 Developing the core application

The core part of the home application is considered to be in a beta stage. It is quite functional, but still needs some more work before it can be left in the hand of the regular user. Most notably, the GUI needs some refinement. A help system should be in place and an automatic update function should be implemented. There are built in functions in the Eclipse framework to provide this functionality, so it should not be a major issue to implement them.

4.3.2 Access control and billing

As described in Part III several ideas as to how a system like this could be restricted and charged was discussed. Most notably an administrator in need of a user based access and billing system would likely generate a separate system for user administration. This would give full flexibility when it comes to billing and other forms of payment.

4.3.3 Security

The XMPP protocol, which is used between each of the entities in this system, supports robust security through Simple Authentication and Security Layer (SASL) with the *external mechanism*, specified in RFC 4422 [150] and using TLS for channel encryption. Both the Smack library, used in the home application and the Openfire XMPP server supports this security scheme; hence it should be quite possible to implement strong security in the system. To encourage the use of secure communications, the XMPP standards foundation XSF runs an intermediate certification authority, giving out free certificates to XMPP server administrators. At this stage in the development, the system does not implement encryption and only password authentication is used on the server and the clients.

4.3.4 SMS interface

During the early stages of the thesis, an SMS gateway solution was discussed. It was considered interesting to be able to search or manipulate the service platform by the use of simple text messages from a mobile phone. Several solutions were discussed with two different architectures in mind. The first solution was to connect a GSM dongle to the home application, and use this as an interface to send and receive text messages to the user. The user would have a separate subscription for the dongle and a private number on which it could be reached. The second solution would utilize a SMS gateway solution where the user sends an SMS to a central number, with a keyword. The gateway would forward the text message to the user's system via the Internet.

4.4 Conclusion

The main goal of this thesis was to make a prototype of a modular service platform and provide connectivity allowing both services and users to be mobile. To highlight and test the functionality of the service platform, an example service using the platform was required.

The thesis has produced a working prototype of a modular service platform, enabling services to be added as plug-ins. The service platform is divided in two parts; one part enabling connectivity using a third party solution over XMPP, and one part enabling a modular framework to add services as plug-ins through Eclipse RCP. The functionality was tested by creating an example search service. It was developed as a plug-in using the modular service platform. The service platform and the example service were tested using a PC with the search service on one side, and a mobile device using Opera Mobile 9.5 on the other.

The service platform does not limit the connectivity to a single network; it provides a path to a service anywhere in the world, as long as a relationship is established. The users access the framework from a web browser; this does not limit the users to a specific platform but lets them access the service from any Internet enabled device.

Located at set-top-boxes or PCs in the home network, an Eclipse RCP application has been created to enable services to access the service platform without connectivity concerns. Eclipse RCP, based on OSGi, enables dynamic activation of services at runtime and allows development of services to be created as plug-ins.

The search service was made to showcase the functionality of the framework, and to highlight unforeseen issues in the service platform. The remote network search is based on the Apache Lucene and Apache Tika projects to provide content-based file search. To enable file transfer, an XMPP client was implemented on the Network Server. The generation of a real world service has proven useful to highlight some of the challenges of creating such a service platform.

Creating a generic framework which works in all network settings is quite a challenge. The proposed solution relies on a stable IETF standard, the XMPP protocol to achieve connectivity, and builds additional complexity on top of it to fit the needs of the framework.

Mobile Remote LAN Master Thesis

References and Appendix

The last part of the report consists of references and appendixes. These documents explain system specifics in more detail, and have references in the report where it is expected.

The reference list uses the ISO 690 - Numerical Reference standard of showing citations.

5 References

[1] **O'Reilly, Tim.** What Is Web 2.0. [Internett] September 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

[2] Make Room, Wikipedia: Internet-based Collaboration Could Change the Way We Do Business. [Internett] February 2007. http://knowledge.wharton.upenn.edu/article.cfm?articleid=1663.

[3] How Internet-Enabled Appliances Can Save You Time & Money. [Internett] September 2008. http://gigaom.com/2008/09/23/how-internet-enabled-appliances-can-save-you-time-money/.

[4] *XMPP based Health Care Integrated Ambient Systems Middleware.* s.l. : Springer Paris, 2008. ISBN: 978-2-287-78543-6.

[5] **Tentori, Mónica, Favela, Jesús og González, Victor M.** Quality of Privacy (QoP) for the Design of Ubiquitous Healthcare Applications. [Internett] 2006. http://www.jucs.org/jucs_12_3/quality_of_privacy_for/jucs_12_03_0252_0269_tentori.html.

[6] Isode Whitepapers: In-depth analysis from Isode. [Internett] http://www.isode.com/whitepapers/white-xmpp.html.

[7] *Communication and mobility behaviour – a trend and panel analysis of the correlation between mobile phone use and mobility.* **Nobis, Claudia and Lenz, Barbara.** 2009, Journal of Transport Geography, pp. 93-103.

[8] *http://www.ibm.com/ibm/cloud/.* [Internett] ftp://ftp.software.ibm.com/software/tivoli/brochures/IBM_Perspective_on_Cloud_Computing.pdf.

[9] *DesktopTwo.* [Internett] http://desktoptwo.com/.

[10] Home Automation - For your life and family. [Internett] HomeSeer. http://www.homeseer.com/.

[11] Xanboo - Bee there, even if you're not. [Internett] Xanboo. http://www.xanboo.com/.

[12] 4Home - Home Control Services for the Connected Home. [Internett] 4Home. http://www.4home.com/.

[13] Bulogics - Simple, green solutions. [Internett] Bulogics. http://www.bulogics.com/.

[14] Power Inside - Energy for your connection. [Internett] Telsey. http://www.telsey.com/homepage.asp.

[15] Homemanageables - Your life, only easier. [Internett] Homemanageables. http://www.homemanageables.com/.

[16] Bobo - Equipping you for the future. [Internett] BoBo Technologies. http://www.bobotechnologies.com/.

[17] Linutop. [Internett] Linutop. http://www.linutop.com/linutop2/index.en.html.

[18] Aaeon - Computing platform service partner. [Internett] Aaeon. http://www.aaeon.com/.

[19] **Callon, R. og Suzuki, M.** A Framework for Layer 3 - Provider-Provisioned Virtual Private Networks (PPVPNs). [Internett] IETF, July 2005. http://www.ietf.org/rfc/rfc4110.txt. RFC4110.

[20] *http://openvpn.net/.* [Internett] OpenVPN. http://openvpn.net/index.php/about/openvpn-facts.html.

[21] **Frankel, Sheila, et al.** Guide to IPsec VPNs. [Internett] December 2005. http://csrc.nist.gov/publications/nistpubs/800-77/sp800-77.pdf. NIST Special Publication 800-77.

[22] **Frankel, Sheila, et al.** Guide to SSL VPNs. [Internett] July 2008. http://csrc.nist.gov/publications/nistpubs/800-113/SP800-113.pdf. NIST Special Publication 800-113.

[23] **Barken, Lee.** *How Secure is Your Wireless Network?* s.l. : Prentice Hall PTR, 2003. ISBN: 978-0131402065.

[24] [Internett] Soonr. http://www.soonr.com/.

[25] [Internett] Meebo. http://www.meebo.com/.

[26] Amazon Elastic Compute Cloud (Amazon EC2). [Internett] Amazone. http://aws.amazon.com/ec2/.

[27] Amazon Simple Storage Service (Amazon S3). [Internett] Amazon. http://aws.amazon.com/s3/.

[28] Isode's Presence, Real Time Messaging and XMPP Strategy. [Internett] Isode. www.isode.com/whitepapers/xmpp.html.

[29] Eclipse RCP MP3 Manager. [Internett] Eclipse. http://max-server.myftp.org/trac/mp3m.

[30] Azureus Bittorrent client. [Internett] Azureus. http://azureus.sourceforge.net/.

[31] Lotus Expeditor enables client integration. [Internett] IBM. http://www-01.ibm.com/software/lotus/products/expeditor/.

[32] **Labidi, Wael, et al.** *XMPP based Health Care Integrated Ambient Systems Middleware.* s.l. : Springer Paris, 2008. ISBN: 978-2-287-78543-6.

[33] **Andreassen, Kristian.** The reliability of XMPP for file transfer. [Internett] June 2008. http://www.ub.uit.no/munin/bitstream/10037/1755/1/thesis.pdf.

[34] Arctis. [Internett] NTNU. http://arctis.item.ntnu.no/.

[35] ISIS. [Internett] NTNU. http://isisproject.org/.

[36] Program for advanced telecom services. [Internett] Telenor. http://www.pats.no/.

[37] **Wack, John, Cutler, Ken og Pole, Jamie.** Guidelines on Firewalls and Firewall Policy. [Internett] January 2002. http://csrc.nist.gov/publications/nistpubs/800-41/sp800-41.pdf. NIST Special Publication 800-41.

[38] **Stallings, William.** *Cryptography and Network Security, 4th Edition.* s.l. : Prentice Hall, 2005. ISBN-13: 978-0-13-187316-2.

[39] **Scarfone, Karen og Hoffman, Paul.** Guidelines on Firewalls and Firewall Policy (Draft). [Internett] July 2008. http://csrc.nist.gov/publications/drafts/800-41-Rev1/Draft-SP800-41rev1.pdf. NIST Special Publication 800-41, Revision 1 (Draft).

[40] **Egevang, K. og Francis, P.** The IP Network Address Translator (NAT). [Internett] May 1994. http://www.ietf.org/rfc/rfc1631.txt. rfc1631.

[41] **Schwartz, David og Sterman, Baruch.** NAT Traversal in SIP. [Internett] September 2005. http://kayote.com/web/docs/WhitePapers/KayoteNetworksWhitePaper-NAT_Traversal_in_SIP.pdf.

[42] **Holzner, Steve.** *Ajax Bible.* s.l. : Wiley, 2007. ISBN: 978-0-470-10263-3.

[43] Introducing Ajax and OpenAjax. [Internett] OpenAjax Alliance. http://www.openajax.org/whitepapers/Introducing%20Ajax%20and%20OpenAjax.php.

[44] **Garrett, Jesse James.** Ajax: A New Approach to Web Applications. [Internett] February 2005. http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[45] **Goodman, Danny og Eich, Brendan.** JavaScript Bible, 6th Edition. [Internett] April 2007. ISBN: 978-0-470-06916-5.

[46] [Internett] ECMA. http://www.ecma-international.org/.

[47] Core JavaScript 1.5 Guide. [Internett] March 2009. https://developer.mozilla.org/En/Core_JavaScript_1.5_Guide.

[48] ECMAScript Language Specification. [Internett] December 1999. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf. Standard ECMA-262, 3rd Edition.

[49] **Flanagan, David.** *JavaScript: The Definitive Guide, 5th Edition.* s.l. : O'Reilly, 2006. ISBN:

9780596101992 | 0596101996.

[50] The XMLHttpRequest Object, W3C Working Draft. [Internett] April 2008. http://www.w3.org/TR/XMLHttpRequest/. WD-XMLHttpRequest-20080415.

[51] Cascading Style Sheets, level 1 - W3C Recommendation. [Internett] April 2008. http://www.w3.org/TR/REC-CSS1/. REC-CSS1-20080411.

[52] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Candidate Recommendation. [Internett] April 2009. http://www.w3.org/TR/CSS21/. CR-CSS2-20090423.

[53] CSS Mobile Profile 2.0, W3C Candidate Recommendation. [Internett] December 2008. http://www.w3.org/TR/css-mobile/. CR-css-mobile-20081210.

[54] PHP. [Internett] PHP Group. http://www.php.net/.

[55] Microsoft asp.net . [Internett] Microsoft. http://www.asp.net/.

[56] JavaServer Pages Technology. [Internett] Sun. http://java.sun.com/products/jsp/.

[57] [Internett] WURFL. http://wurfl.sourceforge.net/.

[58] **Bray, Tim, et al.** Extensible Markup Language (XML) 1.0 (Fifth Edition). [Internett] November 2008. http://www.w3.org/TR/REC-xml/. REC-xml-20081126.

[59] **Clark, James.** XSL Transformations (XSLT), Version 1.0. [Internett] November 1999. http://www.w3.org/TR/xslt. REC-xslt-19991116.

[60] **Sachoff, Mike.** Smartphones See Solid First Quarter Growth. [Internett] May 2009. http://www.webpronews.com/topnews/2009/05/20/smartphones-see-solid-first-quarter-growth.

[61] S60 platform, 3rd Edition Overview. [Internett] February 2007. www.s60.com/pics/pdf/S60_3rd_Ed_2007.pdf.

[62] [Internett] Konqueror. http://www.konqueror.org/.

[63] S60WebKit. [Internett] http://opensource.nokia.com/projects/S60browser/.

[64] S60 browser devices. [Internett] http://www.s60.com/life/s60phones/browseDevices.do.

[65] iPhone. [Internett] Apple. http://www.apple.com/iphone/.

[66] Mobile web browser usage statistics - search engines, devices. [Internett] 2009. http://www.webdevelopersnotes.com/articles/mobile-web-browser-usage-statistics.php.

[67] **Wagner, Richard.** *Professional iPhone and iPod touch Programming : Building Mobile Safari.*

s.l. : Wiley, 2008. ISBN: 978-0-470-25155-3.

[68] Developer IPhone. [Internett] Apple. http://developer.apple.com/.

[69] The most advanced mobile OS. Now even more advanced. [Internett] Apple. http://www.apple.com/iphone/preview-iphone-os/.

[70] **Foresman, Chris.** JavaScript to get 3x speed boost in iPhone OS 3.0. [Internett] March 2009. http://arstechnica.com/apple/news/2009/03/javascript-to-get-3x-speed-boost-in-iphone-os-30.ars.

[71] BlackBerry Browser, Fundamentals Guide. [Internett] 2008. http://na.blackberry.com/eng/deliverables/5687/BlackBerry_Browser-4.7.0-US.pdf. SWDT432285-432285-1205031032-001.

[72] Opera Mobile. [Internett] Opera. http://www.opera.com/mobile/.

[73] Specs Opera 9.5. [Internett] http://www.opera.com/docs/specs/opera95/.

[74] Acid tests. [Internett] Web Standards Project. http://www.acidtests.org/.

[75] **Helenel.** Opera Mobile 9.7 with Opera Turbo. [Internett] March 2009. http://my.opera.com/operamobile/blog/2009/03/26/opera-mobile-9-7-beta-for-windows-mobile.

[76] [Internett] Skyfire. http://get.skyfire.com.

[77] **Haselton, Todd.** Mobile Browser Showdown: iPhone 3G vs Opera Mobile and SkyFire. [Internett] July 2008. http://blog.laptopmag.com/mobile-browser-showdown-iphone-3g-vs-opera-mobile-and-skyfire.

[78] Adobe Flash Player. [Internett] Adobe. http://www.adobe.com/products/flashplayer/.

[79] Microsoft Silverlight. [Internett] Microsoft. http://silverlight.net/.

[80] *QuickTime 7.* [Internett] Apple. http://www.apple.com/quicktime/.

[81] Mozilla Wiki - Fennec. [Internett] https://wiki.mozilla.org/Fennec.

[82] [Internett] Jabber. http://www.jabber.org/.

[83] [Internett] XMPP. http://xmpp.org/.

[84] History of XMPP. [Internett] January 2008. http://xmpp.org/about/history.shtml.

[85] Extensible Messaging and Presence Protocol (XMPP): Core. [Internett] October 2004. http://www.ietf.org/rfc/rfc3920.txt. rfc3920.

[86] Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. [Internett] October 2004. http://www.ietf.org/rfc/rfc3921.txt. rfc3921.

[87] Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM). [Internett] October 2004. http://www.ietf.org/rfc/rfc3922.txt. rfc3922.

[88] End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). [Internett] October 2004. http://www.ietf.org/rfc/rfc3923.txt. rfc3923.

[89] **Day, M., et al.** Instant Messaging / Presence Protocol Requirements. [Internett] February 2000. http://www.ietf.org/rfc/rfc2779.txt. rfc2779.

[90] **Myers, J.** Simple Authentication and Security Layer (SASL). [Internett] October 1997. http://www.ietf.org/rfc/rfc2222.txt. RFC 2222.

[91] **Paterson, Ian, et al.** Bidirectional-streams Over Synchronous HTTP (BOSH). [Internett] April 2009. http://xmpp.org/extensions/xep-0124.html. XEP-0124.

[92] **Paterson, Ian og Saint-Andre, Peter.** XMPP Over BOSH. [Internett] October 2008. http://xmpp.org/extensions/xep-0206.html. XEP-0206.

[93] OSGi™ - The Dynamic Module System for Java™. [Internett] http://www.osgi.org/Main/HomePage.

[94] POJO. [Internett] http://felix.apache.org/site/ipojo-concepts-overview.html.

[95] Equinox. [Internett] http://www.eclipse.org/equinox/.

[96] Rich Client Platform. [Internett] http://wiki.eclipse.org/index.php/Rich_Client_Platform.

[97] **Manning, Christopher D., Raghavan, Prabhakar og Schütze, Hinrich.** *Introduction to Information Retrieval.* s.l. : Cambridge University Press, 2008. ISBN: 0521865719.

[98] **Richardson, Clay W., et al.** *Professional Portal Development with Open Source Tools: Java Portlet API, Lucene, James, Slide.* s.l. : Wiley, 2004. ISBN: 978-0-471-46951-3.

[99] forward index. [Internett] February 2009. http://www.itl.nist.gov/div897/sqg/dads/HTML/forwardIndex.html.

[100] Inverted index. [Online] February 2009. http://www.itl.nist.gov/div897/sqg/dads/HTML/invertedIndex.html.

[101] **Zhang, Jiangong, Long, Xiaohui og Suel, Torsten.** Performance of Compressed Inverted List Caching in Search Engines. [Internett] April 2008. http://www2008.org/papers/pdf/p387-zhangA.pdf.

[102] Inverted file index. [Internett] February 2009. http://www.itl.nist.gov/div897/sqg/dads/HTML/invertedFileIndex.html.

[103] Full inverted index. [Internett] February 2009. http://www.itl.nist.gov/div897/sqg/dads/HTML/fullInvertedIndex.html.

[104] **Shipman, Frank.** IR Models - Overview, Boolean and Vector. [Internett] March 2009. http://www.csdl.tamu.edu/~shipman/courses/cpsc315/slides/vector-model.ppt.

[105] Lucene. [Internett] Apache. http://lucene.apache.org/.

[106] **Hatcher, Erik og Gospodnetic, Otis.** *Lucene in Action - A guide to the Java search engine.* s.l. : Manning Publications Co, 2005. ISBN-13: 978-1-932394-28-3.

[107] Lucene, Class similarity 2.4.0 API. [Internett] http://lucene.apache.org/java/2_4_0/api/org/apache/lucene/search/Similarity.html.

[108] Lucene score. [Internett] http://lucene.apache.org/java/2_3_2/scoring.html.

[109] Lucene - Index File Formats. [Internett] http://lucene.apache.org/java/2_4_0/fileformats.html.

[110] **Zhou, Deng Peng.** Delve inside the Lucene indexing mechanism. [Internett] June 2006. http://www.ibm.com/developerworks/library/wa-lucene/#Listing1.

[111] Lucene - Query Parser Syntax. [Internett] http://lucene.apache.org/java/2_4_0/queryparsersyntax.html.

[112] **Spolsky, Joel.** The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets. [Internett] October 2003. http://www.joelonsoftware.com/articles/Unicode.html.

[113] Martin Porter's Home Page. [Internett] http://tartarus.org/~martin/index.html.

[114] The Porter Stemming Algorithm. [Internett] http://tartarus.org/~martin/PorterStemmer/.

[115] Lucene - Lucene Sandbox. [Internett] http://lucene.apache.org/java/2_4_0/lucene-sandbox/index.html.

[116] Stemming algorithms for various European languages. [Internett] http://snowball.tartarus.org/texts/stemmersoverview.html.

[117] Tika - Supported Document Formats. [Internett] http://lucene.apache.org/tika/formats.html.

[118] Apache server project. [Internett] Apache. http://httpd.apache.org/.

[119] HTTPD - Apache2 Web Server. [Internett] Ubuntu. https://help.ubuntu.com/8.10/serverguide/C/httpd.html.

[120] **Moffitt, Jack.** Which BOSH Server Do You Need? [Internett] September 2008. http://metajack.im/2008/09/08/which-bosh-server-do-you-need/.

[121] Tigase XMPP/Jabber Server. [Internett] Tigase. http://www.tigase.org/.

[122] Ejabbered - the Erlang Jabber/XMPP daemon. [Internett] Ejabbered. http://www.ejabberd.im/.

[123] Openfire - XMPP Server. [Internett] Ignite. http://www.igniterealtime.org/projects/openfire/index.jsp.

[124] Database Installation guide. *Igniterealtime community.* [Internett] Jive Software - Igniterealtime community, 2009. http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/database.html.

[125] *Knopflerfish OSGi framework.* [Internett] http://www.knopflerfish.org/.

[126] *Apache Felix Project R4 OSGi implementation.* [Internett] http://felix.apache.org/.

[127] OpenSolaris Project: CIFS client for Solaris. [Internett] Sun. http://opensolaris.org/os/project/smbfs/.

[128] Google Desktop Search APIs. [Internett] Google. http://code.google.com/intl/nb-NO/apis/desktop/docs/searchapi.html.

[129] Windows Search 4.0 for Developers. [Internett] Microsoft. http://www.microsoft.com/windows/products/winfamily/desktopsearch/choose/windowssearch4/developers.mspx.

[130] Xapian home page. [Internett] Xapian. http://xapian.org/.

[131] Egothor - Java search engine and supplemental libraries. [Internett] Egothor. http://www.egothor.org/.

[132] MSN Web Messenger. [Internett] Microsoft. http://webmessenger.msn.com/.

[133] Google Talk. [Internett] Google. http://www.google.com/talk/.

[134] Meebo - Instant messaging everywhere. [Internett] Meebo. http://www.meebo.com/.

[135] File Transfer on meebo Raises the Bar for Web-Based Instant Messaging. [Internett] September 2007. http://www.meebo.com/press/releases/20070910/.

[136] Apache Module mod_proxy. [Internett] http://httpd.apache.org/docs/2.0/mod/mod_proxy.html.

[137] Apache Module mod_rewrite. [Internett]

http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html.

[138] JSJaC API. [Internett] http://blog.jwchat.org/jsjac-1.3.2/doc/.

[139] Jabber HTTP Polling. [Internett] http://xmpp.org/extensions/xep-0025.html. XEP-0025.

[140] **Clayberg, Eric og Rubel, Dan.** *Eclipse: Building Commercial-Quality Plug-ins (2nd Edition).* s.l. : Addison-Wesley Professional, 2006. ISBN-13: 978-0321426727.

[141] Maven Home Page. [Internett] Apache. http://maven.apache.org/.

[142] XMPP Is Better With BOSH . [Internett] Junly 2008. http://metajack.wordpress.com/2008/07/02/xmpp-is-better-with-bosh/.

[143] IP packet overhead. [Internett] http://www.tamos.net/~rhay/overhead/ip-packet-overhead.htm.

[144] Benchmarking BOSH-Services. [Internett] May 2009. http://dev.esl.eu/blog/tag/xmpp/.

[145] Google Androïd SDK not XMPP compliant. [Internett] February 2008. http://mail.jabber.org/pipermail/standards/2008-February/018015.html.

[146] **Rysavy, Peter.** GPRS to HSDPA and beyond. [Internett] September 2005. http://www.rysavy.com/Articles/Rysavy_Data_Paper-Sept2005.pdf.

[147] Deknings kart Telenor mobil. [Internett] Telenor. http://www.telenor.no/privat/mobil/dekning/dekningskart/.

[148] Dekningskart Netcom. [Internett] Netcom. https://netcom.no/bedrift/kundeservice/abonnement-og-tjenester/dekning.html.

[149] **Deutsch, P.** GZIP file format specification version 4.3. [Internett] May 1996. http://www.ietf.org/rfc/rfc1952.txt. RFC 1952.

[150] **Melnikov, A. og Zeilenga, K.** Simple Authentication and Security Layer (SASL). [Internett] June 2006. http://www.ietf.org/rfc/rfc4422.txt. RFC 4422.

[151] **http://sites.google.com/site/mobileremotelan/Home/filecabinet.** tihgag asdfga sdasg as fsdaf sadf sadf sdf sdf . *asdf asdf asfasdf sdf asgfgawerg.* [Internett] asdf asdf asdf asasdfgadfgdaf, 2 2 2000. http://sites.google.com/site/mobileremotelan/Home/filecabinet.

[152] **Chevul, Stefan, et al.** *Measurement of Application-Perceived Throughput of an E2E VPN Connection Using a GPRS Network.* s.l. : Springer Berlin / Heidelberg, 2006. ISBN: 978-3-540-34025-6.

[153] XMPP Software: Servers. [Internett] XMPP. http://xmpp.org/software/servers.shtml.

[154] **Dhana, Sokol.** custompublish.com. *Short messaging solutions, including XMPP based instant messaging and text based conferences, between health care providers and general practitioners.* [Internett] 2005. http://img.custompublish.com/getfile.php/796147.357.awdevpufde/Practice%20Week/BicerLDK05.pdf.

6 Appendix

6.1 Appendix A – Plug-in development for the Home Server

6.1.1 How to add new services to the framework

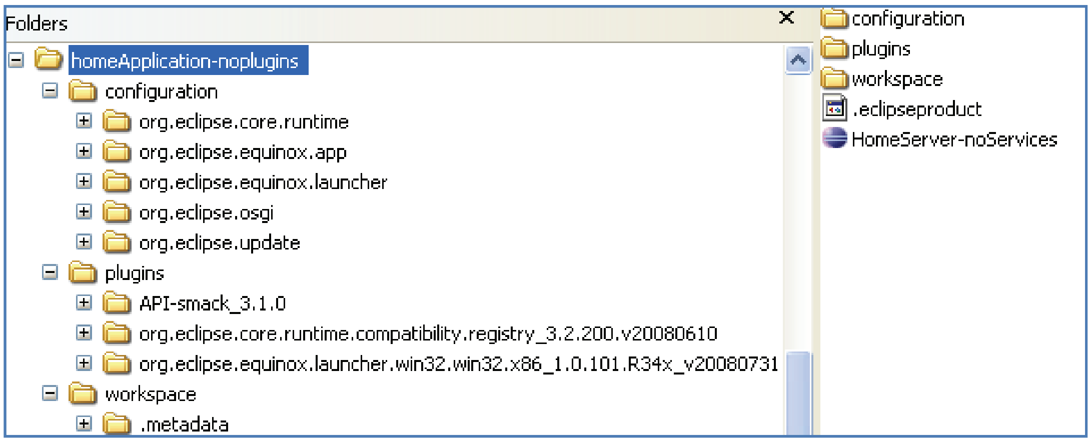
To understand the basics of creating a new service for the home application it is necessary to understand the basic anatomy of the Eclipse framework, as they are essentially the same. As described in earlier chapters, the home application is built by using the basic building blocks of the Eclipse platform and provides the same functionality to its components. There are numerous good books covering the creation of plug-ins for Eclipse [ref. building commercial quality plug-ins for eclipse]. The process of making plug-ins for the home application follows the same guidelines and best practices as used when developing eclipse plug-ins. The impact of using a readymade framework is quite large. Because the home application itself is an eclipse plug-in, it too can be included in a running eclipse platform instead of in a standalone Eclipse RCP application.

6.1.1.1 External requirements

To be able to test new plug-ins for the home application, some prerequisites need to be in place. The hscore plug-in needs to be able to connect to the preconfigured XMPP server over the Internet. The parameters for this server are set in the hscore plug-in. It is also possible to create one's own XMPP server, but then new parameters for the server need to be set in the code of the hscore plug-in. The Openfire server from Jive Software can be recommended [jivesoftware.org]. Configuring this is outside the scope of this tutorial.

6.1.1.2 Getting hold of the core application

This chapter will basically follow the design of a how-to tutorial for getting started with eclipse plug-ins. The Home Application will be distributed as a standalone application. The structure of this application is the same as an eclipse distribution; the difference is the number of plug-ins included. The basic structure is shown in Figure 68



```
Folders
homeApplication-noplugins
    configuration
        org.eclipse.core.runtime
        org.eclipse.equinox.app
        org.eclipse.equinox.launcher
        org.eclipse.osgi
        org.eclipse.update
    plugins
        API-smack_3.1.0
        org.eclipse.core.runtime.compatibility.registry_3.2.200.v20080610
        org.eclipse.equinox.launcher.win32.win32.x86_1.0.101.R34x_v20080731
    workspace
        .metadata

configuration
plugins
workspace
.eclipseproduct
HomeServer-noServices
```

Figure 68 - Home Application basic file structure

To be able to code and debug a new plug-in, it is necessary to have access to the core application. It is important to note that it is not necessary to have the source code of the hscore plug-in. The binary build will do. This is the same build as the one distributed to clients.

6.1.1.3 Setting up the environment

The following example is demonstrated to make it easier for a plug-in developer to get started. Previous experience with the Eclipse IDE is preferred. The following example is performed using the Sun Java SE Runtime Environment "JRE 6"[http://java.sun.com/javase/downloads/index.jsp]; JRE 5 should also work but is not tested. The platform used is the "Eclipse for RCP/Plug-in Developers", which can be downloaded from the eclipse.org [http://www.eclipse.org/downloads/] website. The current version is the Eclipse Ganymede SR2, (version 3.4.2). It is important to realize that since we are developing against the eclipse platform itself, it is not certain this tutorial will be valid for any future release of eclipse, as it is an evolving platform. Even so, the basics are the same.

6.1.1.4 Creating an example plug-in

Once the platform is up and running it is time to import the core application into the running workspace. This enables running and debugging the new plug-in without exporting it.

In your new Eclipse for RCP/Plug-in developers Select File>Import>Plug-ins and Fragments, click next.

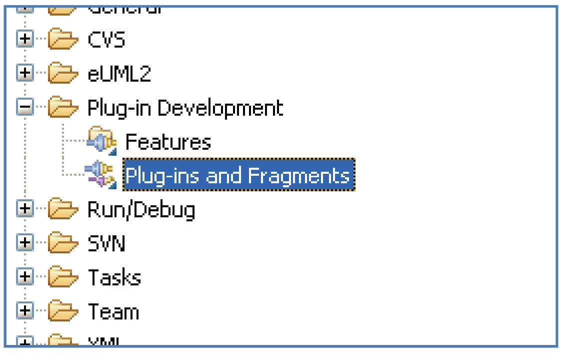
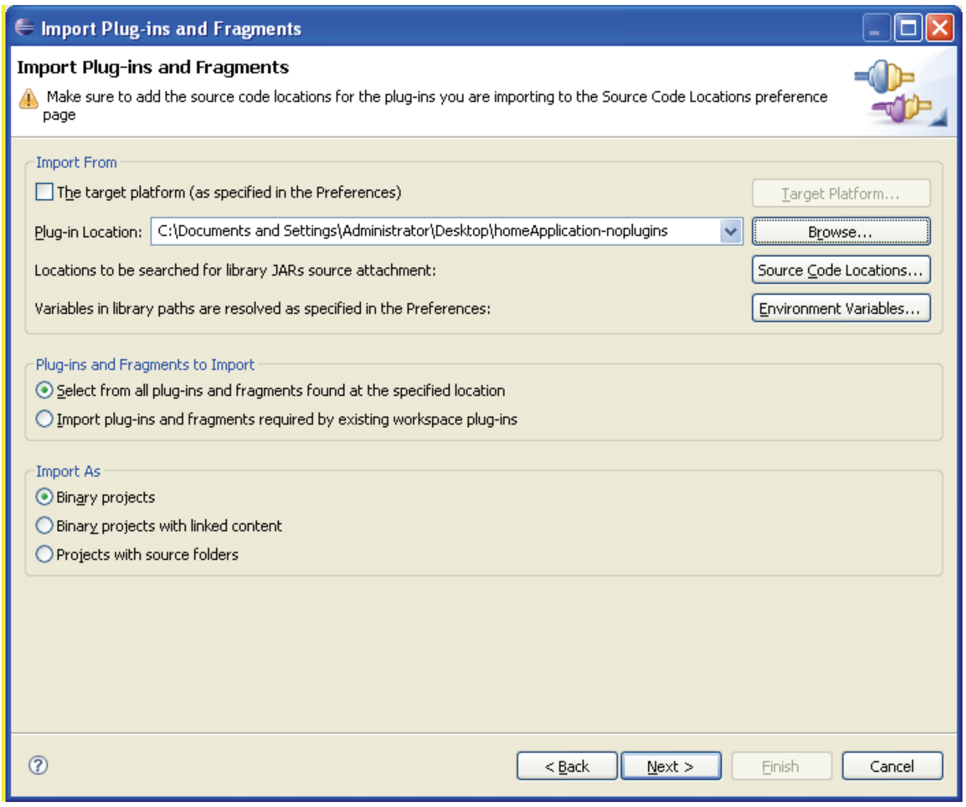


Figure 69 - Import Plug-ins and Fragments 1

In the next page>remove top-left checkbox>click Browse>navigate to the folder of the Home Application>click ok>click next


Import Plug-ins and Fragments

Import Plug-ins and Fragments

Make sure to add the source code locations for the plug-ins you are importing to the Source Code Locations preference page

Import From

☐ The target platform (as specified in the Preferences) — Target Platform...

Plug-in Location: C:\Documents and Settings\Administrator\Desktop\homeApplication-noplugins — Browse...

Locations to be searched for library JARs source attachment: — Source Code Locations...

Variables in library paths are resolved as specified in the Preferences: — Environment Variables...

Plug-ins and Fragments to Import

[✓ Select from all plug-ins and fragments found at the specified location]

Import plug-ins and fragments required by existing workspace plug-ins

Import As

[✓ Binary projects]

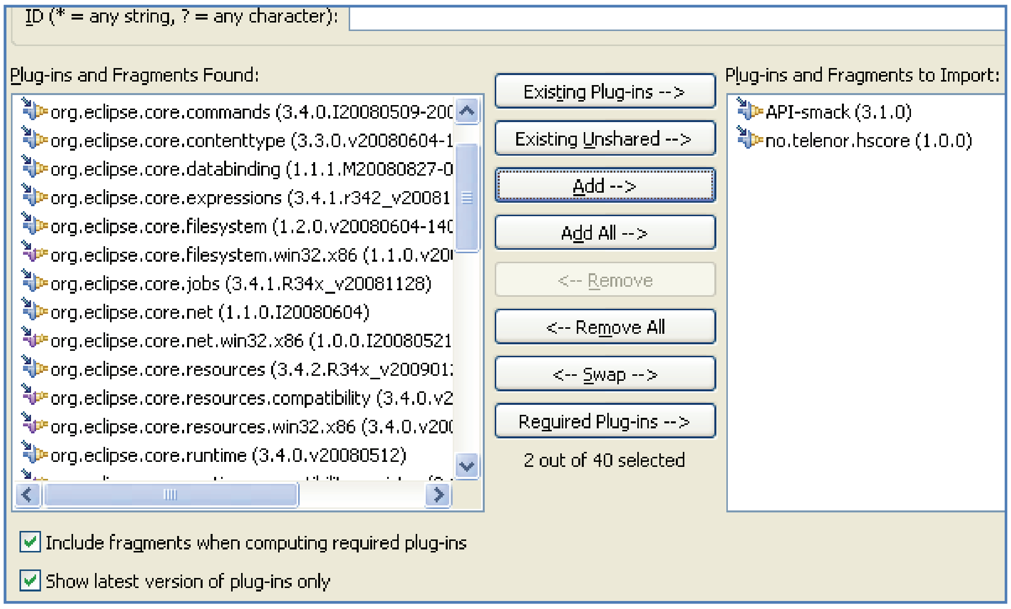
Binary projects with linked content

Projects with source folders

< Back | Next > | Finish | Cancel

Figure 70 - Import Plug-ins and Fragments 2

On the next page, select the plug-in: "API-smack" and "no.telenor.hscore" and click finish


ID (* = any string, ? = any character):

Plug-ins and Fragments Found:

org.eclipse.core.commands (3.4.0.I20080509-200
org.eclipse.core.contenttype (3.3.0.v20080604-1
org.eclipse.core.databinding (1.1.1.M20080827-0
org.eclipse.core.expressions (3.4.1.r342_v20081
org.eclipse.core.filesystem (1.2.0.v20080604-140
org.eclipse.core.filesystem.win32.x86 (1.1.0.v200
org.eclipse.core.jobs (3.4.1.R34x_v20081128)
org.eclipse.core.net (1.1.0.I20080604)
org.eclipse.core.net.win32.x86 (1.0.0.I20080521
org.eclipse.core.resources (3.4.2.R34x_v200901
org.eclipse.core.resources.compatibility (3.4.0.v2
org.eclipse.core.resources.win32.x86 (3.4.0.v200
org.eclipse.core.runtime (3.4.0.v20080512)

Existing Plug-ins -->
Existing Unshared -->
Add -->
Add All -->
<-- Remove
<-- Remove All
<-- Swap -->
Required Plug-ins -->
2 out of 40 selected

Plug-ins and Fragments to Import:

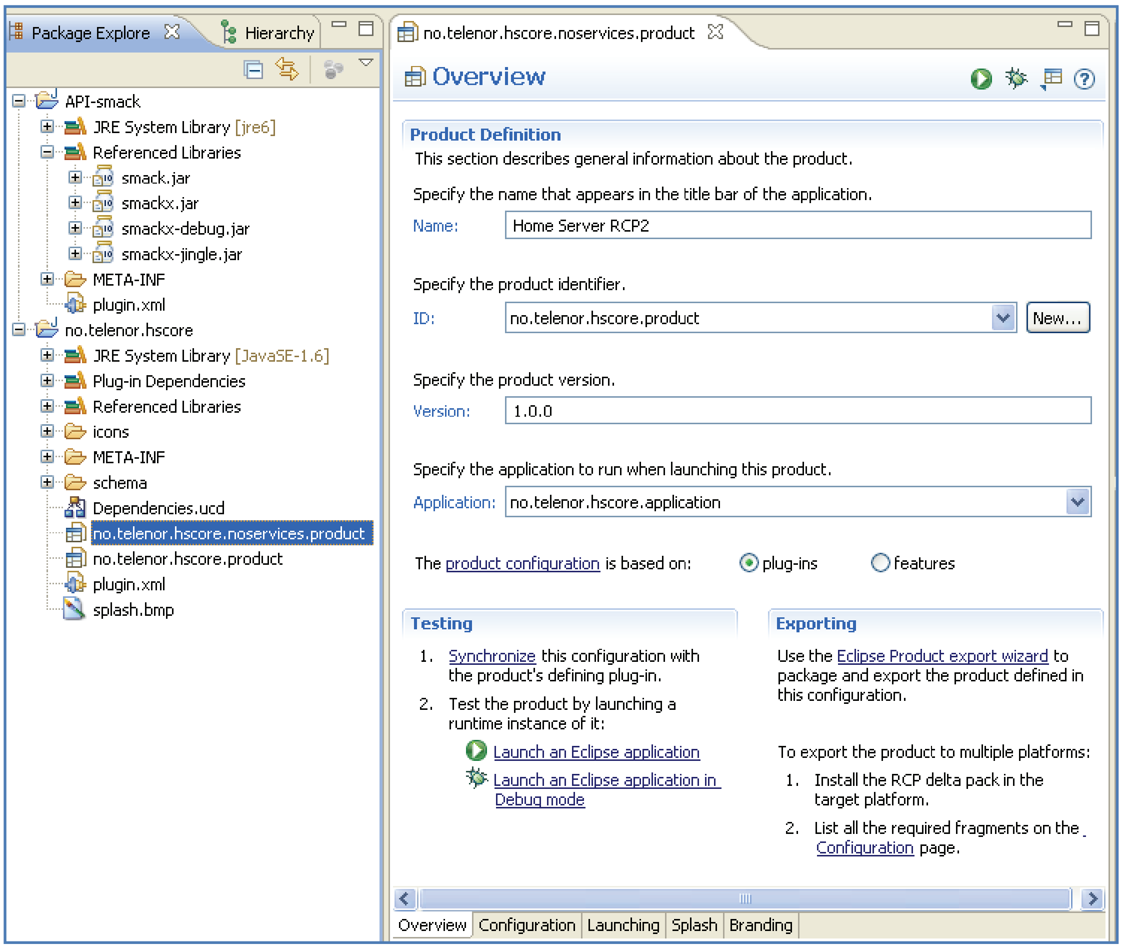
API-smack (3.1.0)
no.telenor.hscore (1.0.0)

☑ Include fragments when computing required plug-ins

☑ Show latest version of plug-ins only

Figure 71 - Import Plug-ins and Fragments 3

Two new projects should now be available in your Package Explorer. The custom libraries are included in the Referenced Libraries folders of each project. The "API-smack" provides XMPP communication functionality, while "no.telenor.hscore" is the base application. It is now possible to test run the hscore application without any services. Go into the project folder, and double click on the no.telenor.hscore.noservices.product file. This file defines an Eclipse product, a configuration made up of several plug-ins and custom branding. Click on the "Launch and Eclipse Application" in the Overview tab. A splash screen is shown and the application is launched. Notice how the output of the application is shown in the eclipse console. Just as when coding a regular java application.



This section describes general information about the product.

Specify the name that appears in the title bar of the application.

Name: Home Server RCP2

Specify the product identifier.

ID: no.telenor.hscore.product New...

Specify the product version.

Version: 1.0.0

Specify the application to run when launching this product.

Application: no.telenor.hscore.application

The product configuration is based on: [✓ plug-ins] features

1. Synchronize this configuration with the product's defining plug-in.
2. Test the product by launching a runtime instance of it:

Launch an Eclipse application

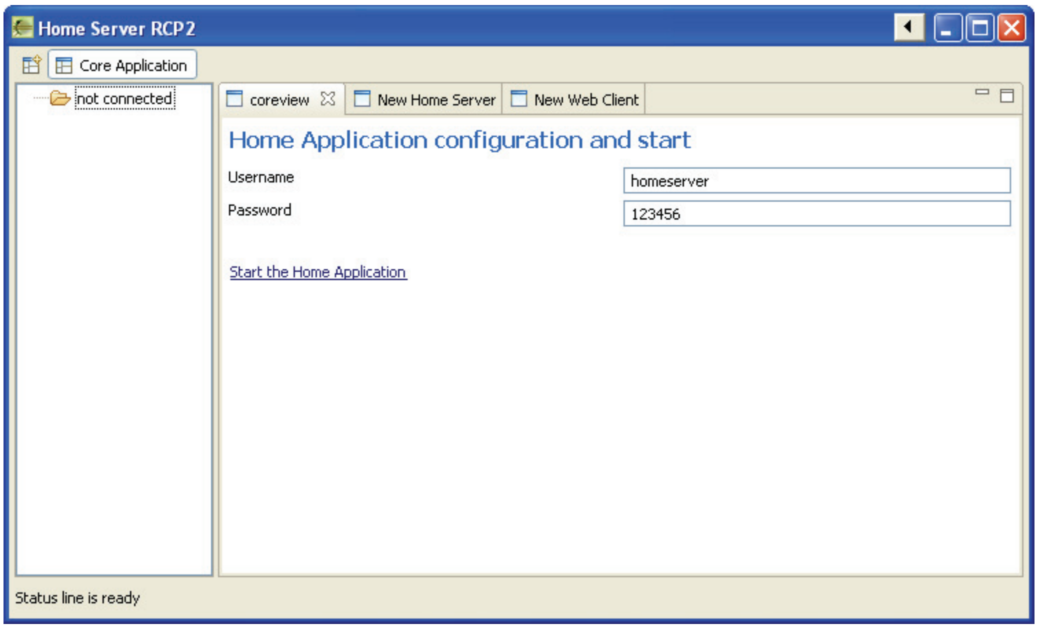
Launch an Eclipse application in Debug mode

Use the Eclipse Product export wizard to package and export the product defined in this configuration.

To export the product to multiple platforms:

1. Install the RCP delta pack in the target platform.
2. List all the required fragments on the Configuration page.

Figure 72 - Dry run of the hscore plug-in



Home Application configuration and start

Username homeserver

Password 123456

Start the Home Application

Status line is ready

Figure 73 - Home Application without any plug-ins

Okay, now the platform is up and running, and it is possible to test the Home Application. The next step is to create a new plug-in which ties into the hscore plug-in, contributing to its GUI and extending its extension point with its own contribution. Go to File>New>Project>Plug-in Development>Plug-in Project>Click Next>

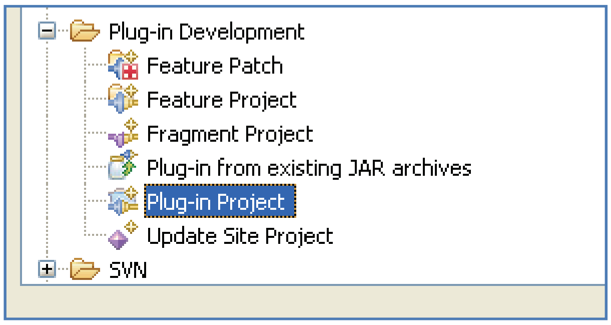
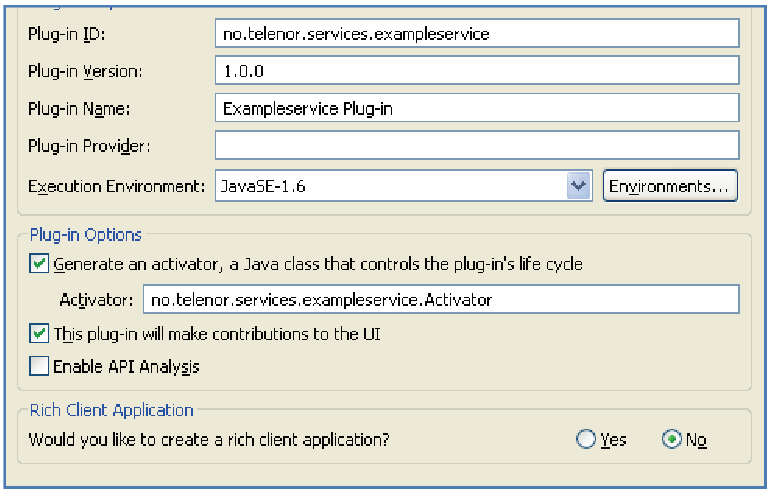


Figure 74 - New Plug-in project

Create a name for the new project. To be detected as a service in the hscore application, it is important the new plug-in conforms to the naming scheme for hscore services. The following is an example of a valid naming scheme; "no.telenor.services.exampleservice" The three first names should be kept static throughout all plug-ins, while the last name is custom. This is because the hscore plug-in keeps a list of all plug-ins conforming to this naming scheme and notifies its clients about these services. If the new plug-in needs additional new plug-ins to function, these should be given a name conforming to "no.telenor.services.newserviceexample.additionalplugin" These will not be listed as services to the clients and can be used for plug-in internal functions. Make sure the Target Platform is set to the same as the eclipse you are running (currently 3.4) and click next. On the next page, make sure the settings are set as shown in Figure 75. The plug-in should make contributions to the UI and it should not be an RCP application. Click next.



Plug-in ID:	no.telenor.services.exampleservice
Plug-in Version:	1.0.0
Plug-in Name:	Exampleservice Plug-in
Plug-in Provider:	
Execution Environment:	JavaSE-1.6

Plug-in Options

✓ Generate an activator, a Java class that controls the plug-in's life cycle

Activator: no.telenor.services.exampleservice.Activator

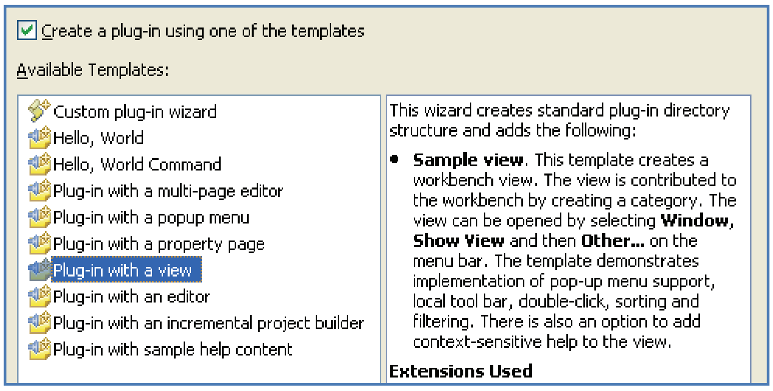
✓ This plug-in will make contributions to the UI

☐ Enable API Analysis

Rich Client Application

Would you like to create a rich client application? Yes [✓ No]

Figure 75 - Creating a new plug-in project



☑ Create a plug-in using one of the templates

Available Templates:

- Custom plug-in wizard
- Hello, World
- Hello, World Command
- Plug-in with a multi-page editor
- Plug-in with a popup menu
- Plug-in with a property page
- Plug-in with a view
- Plug-in with an editor
- Plug-in with an incremental project builder
- Plug-in with sample help content

This wizard creates standard plug-in directory structure and adds the following:

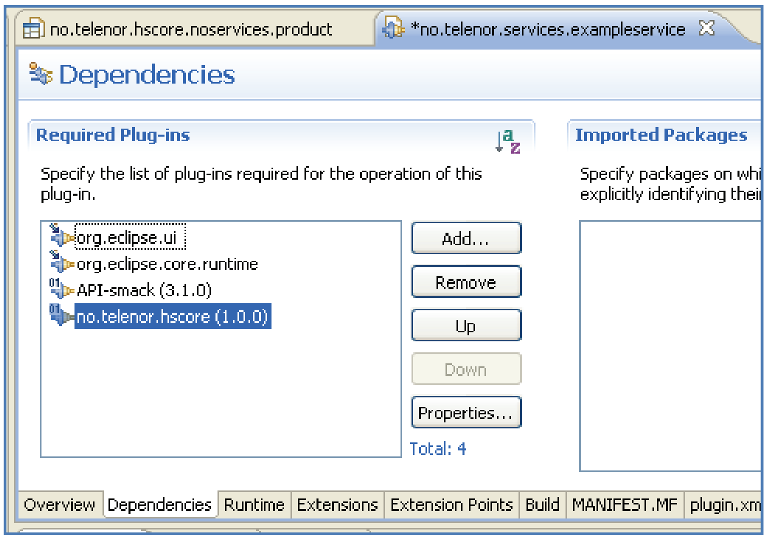
- **Sample view**. This template creates a workbench view. The view is contributed to the workbench by creating a category. The view can be opened by selecting **Window, Show View** and then **Other...** on the menu bar. The template demonstrates implementation of pop-up menu support, local tool bar, double-click, sorting and filtering. There is also an option to add context-sensitive help to the view.

Extensions Used

Figure 76 - Creating a new plug-in project

For this example, choose the Plug-in with a view. Click Finish. If there is a pop-up asking to open the associated perspective, click yes. This will open up the eclipse workbench for plug-in development. In the Package Explorer the new project has popped up. The most important file in any plug-in is the plugin.xml file where most of the configuration tasks are executed; double clicking it will open it up on the Overview page.

The next step is to integrate the new plug-in with the hscore RCP application. Go to the Dependencies tab in the plugin.xml file. Add the plug-ins that constitutes the hscore application, the “no.telenor.hscore” and the “API-smack” to the list of Required Plug-ins as shown in Figure 77. This will make sure this plug-in can use functions inside the other plug-ins.



no.telenor.hscore.noservices.product | *no.telenor.services.exampleservice

Dependencies

Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in.

- org.eclipse.ui
- org.eclipse.core.runtime
- API-smack (3.1.0)
- no.telenor.hscore (1.0.0)

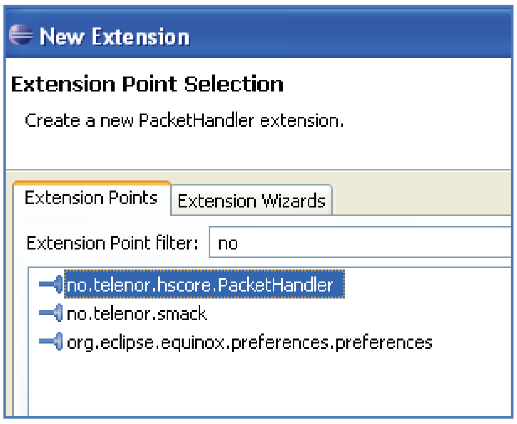
Total: 4

Imported Packages

Specify packages on whi
explicitly identifying thei

Figure 77 - Required plug-ins

The next step is to enable the new plug-in to extend the PacketHandler extension point made available by the no.telenor.hscore plug-in. This will enable the new plug-in to receive messages sent by clients over the network. Go to The extensions tab and add the “no.telenor.hscore.PacketHandler” extension point


New Extension

Extension Point Selection

Create a new PacketHandler extension.

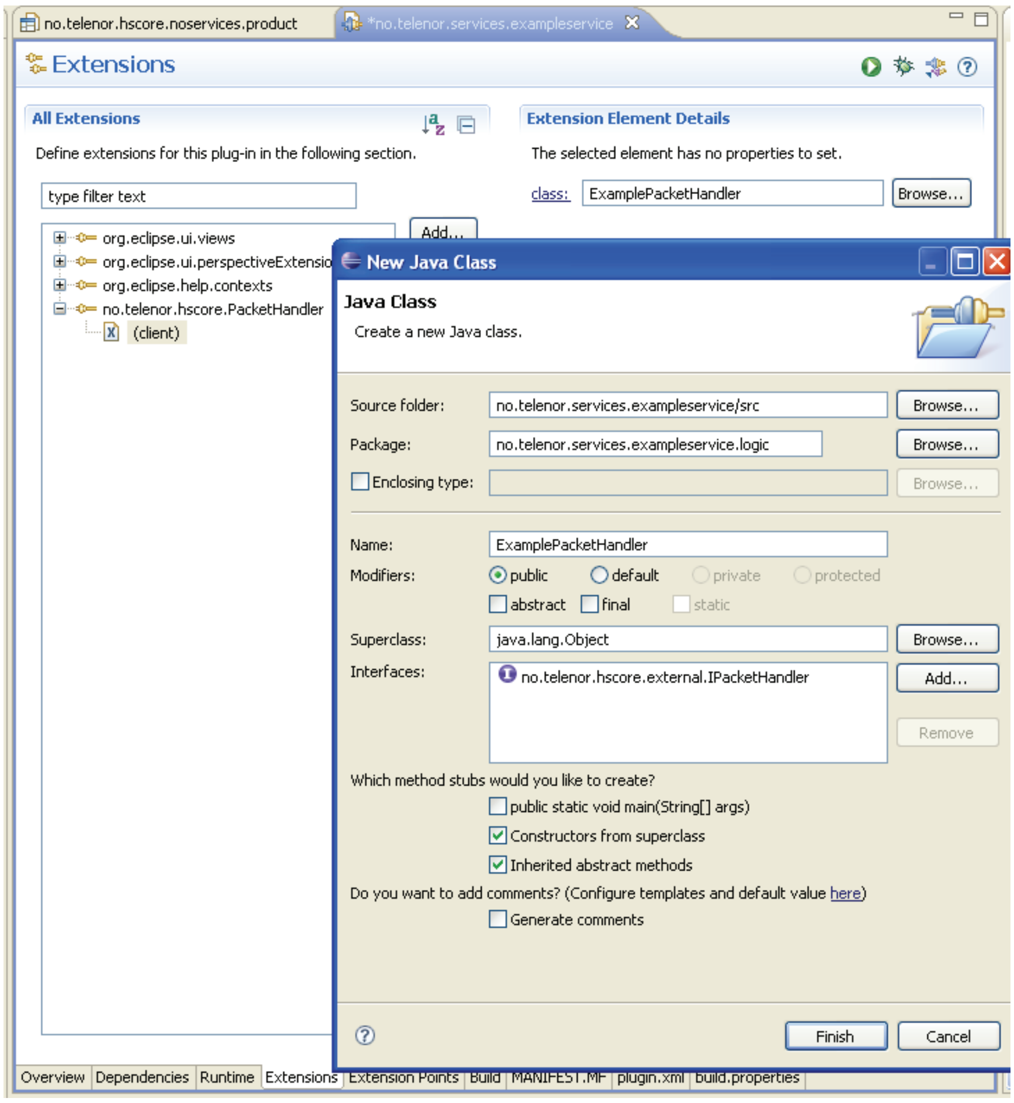
Extension Points | Extension Wizards

Extension Point filter: no

- no.telenor.hscore.PacketHandler
- no.telenor.smack
- org.eclipse.equinox.preferences.preferences

Figure 78 - Add new extension

In the Extensions tab, right click on the new extension and select new>client


no.telenor.hscore.noservices.product | *no.telenor.services.exampleservice

Extensions

All Extensions

Define extensions for this plug-in in the following section.

type filter text

- org.eclipse.ui.views
- org.eclipse.ui.perspectiveExtensio
- org.eclipse.help.contexts
- no.telenor.hscore.PacketHandler
  - (client)

Extension Element Details

The selected element has no properties to set.

class: ExamplePacketHandler

New Java Class

Java Class

Create a new Java class.

Source folder: no.telenor.services.exampleservice/src

Package: no.telenor.services.exampleservice.logic

Enclosing type:

Name: ExamplePacketHandler

Modifiers: [✓ public] default private protected

abstract final static

Superclass: java.lang.Object

Interfaces: no.telenor.hscore.external.IPacketHandler

Which method stubs would you like to create?

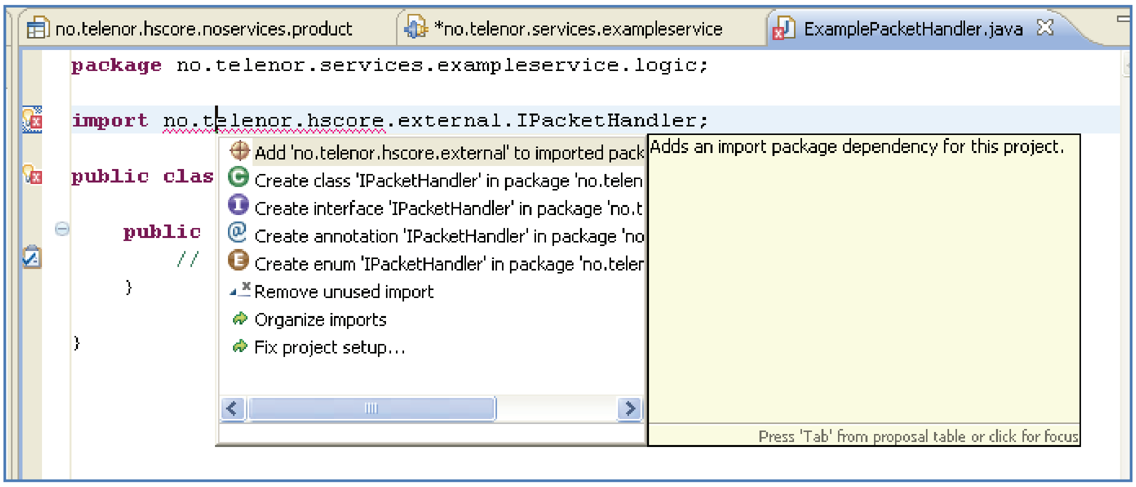
- public static void main(String[] args)
- [✓ Constructors from superclass]
- [✓ Inherited abstract methods]

Do you want to add comments? (Configure templates and default value here)

- Generate comments

Figure 79 - Add new client to extension point PacketHandler and create class for the client

Next, type in a class name for the new class e.g. ExamplePacketHandler, then click on the “class:” link. This will produce a new class wizard. All the fields are already made; the class name and its interface class the IPacketHandler from the no.telenor.hscore plug-in. Clicking finish will open the new class.



```
package no.telenor.services.exampleservice.logic;

import no.telenor.hscore.external.IPacketHandler;

public clas
    public
        //
    }
}
```

Add 'no.telenor.hscore.external' to imported pack
Create class 'IPacketHandler' in package 'no.telen
Create interface 'IPacketHandler' in package 'no.t
Create annotation 'IPacketHandler' in package 'no
Create enum 'IPacketHandler' in package 'no.teler
Remove unused import
Organize imports
Fix project setup...

Adds an import package dependency for this project.

Press 'Tab' from proposal table or click for focus

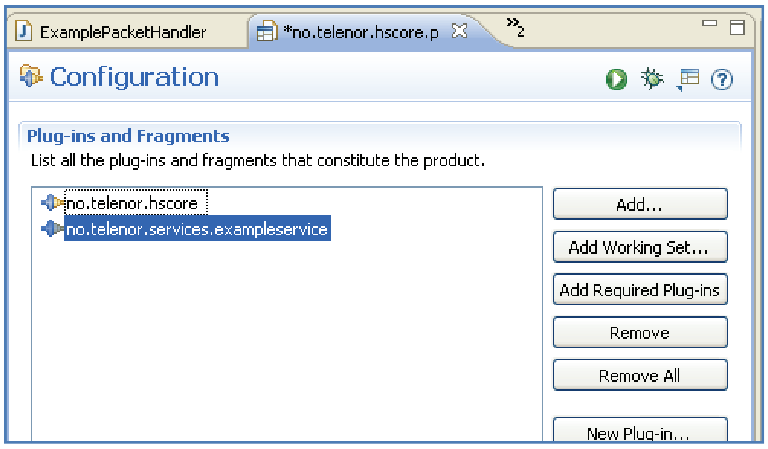
no.telenor.hscore.noservices.product
*no.telenor.services.exampleservice
ExamplePacketHandler.java

Figure 80 - Newly created ExamplePacketHandler

As seen, the class has two errors; the first one is solved by clicking on the error and adding the no.telenor.hscore.external package to the imported packages list, while the second is solved by adding the unimplemented method “incommingPacket” from the implemented interface IPacketHandler defined in no.telenor.hscore.external package. In the below code, some simple print lines are added to the new method to show us when and what is displayed when a packet is received.

```
@Override
        public void incommingPacket(Document arg0, XMPPConnection arg1, Packet arg2) {
                // TODO Auto-generated method stub
                System.out.println("*************************************");
                System.out.println("ExamplePacketHandler was triggered");
                System.out.println("packet came from: "+arg2.getFrom());
                System.out.println("this user is: "+arg1.getUser());
                System.out.println("The packet content: "+arg2.toXML());
                System.out.println("*************************************");
        }
```

The next step is to run this new plug-in in the context of the hscore product configuration. To do this, go to the Package Explorer, open the no.telenor.hscore project and double click on the no.telenor.hscore.product file. In an Eclipse RCP setting, a product is a collection of plug-ins which constitutes a running application. Go to the Configuration tab, here all the plug-ins and features for the hscore product is found. Click the remove all button on the right hand side. Then add the no.telenor.hscore and no.telenor.services.exampleservice as shown in Figure 81.



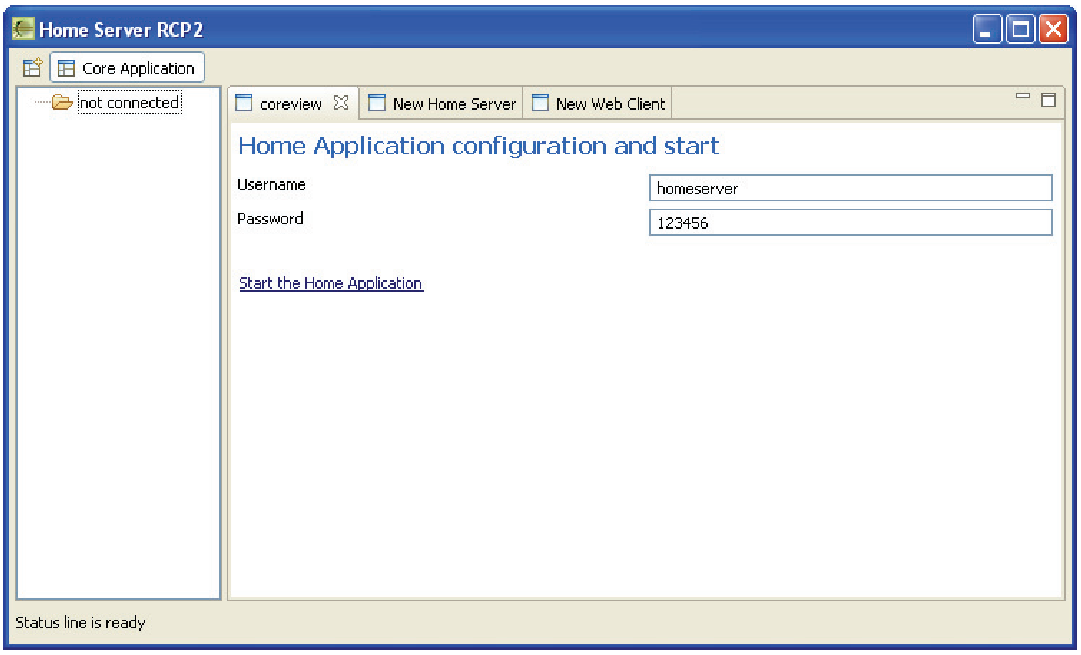
Plug-ins and Fragments

List all the plug-ins and fragments that constitute the product.

no.telenor.hscore
no.telenor.services.exampleservice

Figure 81 - hscore product configuration, adding plug-ins

Next, click the Add Required Plug-ins button. Notice how the API-smack plug-in and more than 30 other plug-ins were added automatically because the two plug-ins already added require the presence of these secondary plug-ins to operate. Now everything should be ready to do a first build and run of the application. Go to the overview tab of the no.telenor.hscore.product file and click "Launch an Eclipse application. A splash screen should display for a second, and then the main window should be displayed as shown in Figure 82. Observe that we have not made any attempt to add any GUI component to our exampleplugin, hence it will not be shown here yet. What we have done is to create a small plug-in which should react once we send a packet to it.



Home Application configuration and start

Username: homeserver
Password: 123456

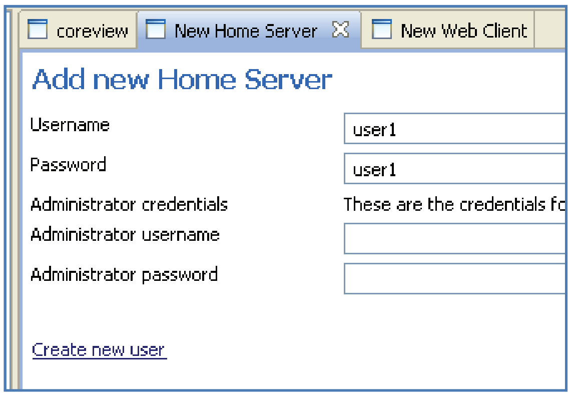
Status line is ready

Figure 82 - Home Server Main window

6.1.1.5 Setting up a communications framework

The Hscore plug-in uses the XMPP communications protocol to relay information between users registered with the network server. The easiest way to debug and develop our own communications on top of this is to install a simple XMPP chat client to which we communicate when creating our new plug-in. This tutorial uses the Open Source "Spark IM client" version 2.5.8 from JiveSoftware[http://www.igniterealtime.org/projects/spark/index.jsp], the same company responsible for the "Openfire" XMPP server used in this assignment. The selection of IM client is totally irrelevant to the task at hand; any XMPP compliant client will do the trick. Install it on the developer computer before proceeding.

The next task is to create a couple of users to use when communicating. In the started home server app, go to the "New Home Server" tab and create a username and password to use with the new home server. This will be considered the "Admin" credentials later on when associating new users to this home servers group of friends. Make sure to click "Create New User" at the bottom. When running the application from inside Eclipse, the debug information should show up inside the eclipse console.

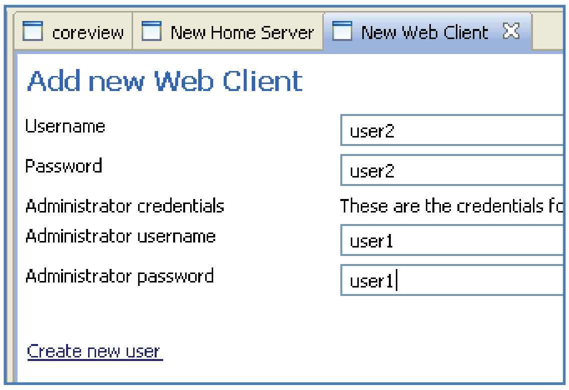


Add new Home Server

Username	user1
Password	user1
Administrator credentials	These are the credentials fc
Administrator username	
Administrator password	

Figure 83 - Create a new Home Server

Next, switch to the "new Web Client" tab and type in the credentials for another user.



Add new Web Client

Username	user2
Password	user2
Administrator credentials	These are the credentials fc
Administrator username	user1
Administrator password	user1

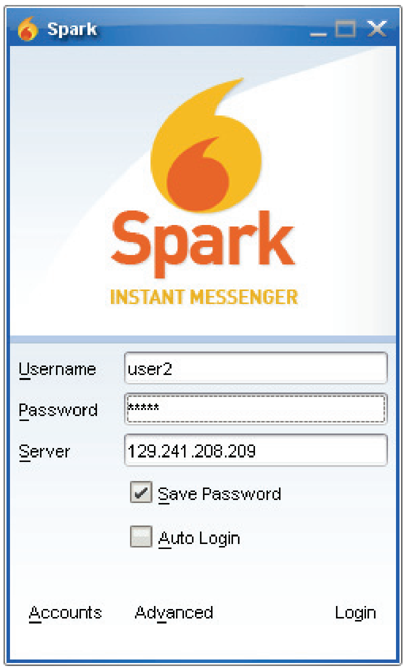
Figure 84 - create a new Client user

Make sure to type in the user1 credentials in the Administrator credentials field. This will associate user 2 to user 1 by adding it to the roster, or friends list of user 1. This way of adding a user is somewhat different from a regular IM application where user1 interactively accept the new friend request. Here this is handled in the background, so that when the "Create new user" button is clicked, the new user gets a mutual authentication with user1 by logging in as user1 and accepting the request.

Once this is set up, go back to the core view tab, type in the credentials for user1 and click "Start the Home Application" Notice how the console in eclipse prints some debugging information to the screen. The below text is a selection of the printouts. The first part is important, here it is possible to see all the registered plug-ins in the OSGi framework. At this point, only one plug-in of service type is registered. These are the services added to the servicesList, which can be fetched by clients if they send a getServices packet to user1. The last section in the printout shows user1's friends and their status. "both" is the status of the subscription and means both sides of the connection have acknowledged the association.

```
******************************************************************************
*                        ** Adding available services **                     *
******************************************************************************
ExtensionPointLabel              : PacketHandler
contributor name                 : no.telenor.services.exampleservice
Service added to servicesList    : exampleservice
***************************Finished adding services**************************
+++rosterUpdater
new roster created:
rosterCreator, sending rosterGUIupdate
**********Contents of rosterStore: 1 Users*******
* user2@129.241.208.209
* unavailable
* both
* WEB
*********
**************************************************
```

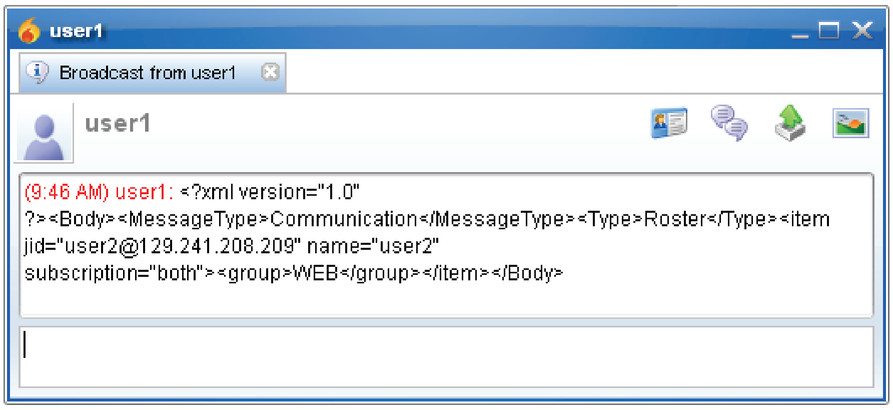
Let's now start up the Spark IM client, which will be used to send and receive messages to our new application. Type in the user2 credentials, the correct server and log in.



Username user2
Password *****
Server 129.241.208.209
[✓] Save Password
[ ] Auto Login

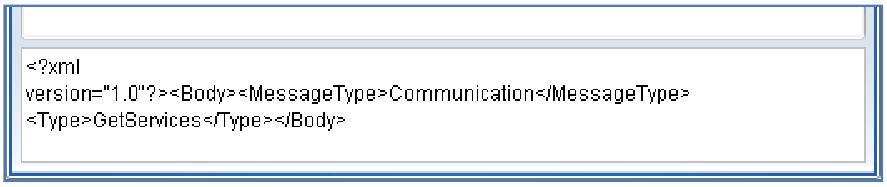
Figure 85 - Spark IM client for debugging

Once logged in, the hscore plug-in in the new application sends a message packet to the logged on user containing the roster entries of user1. This feature is included to let the new user add these users to his roster and get a roster with the same content as user1. We will now send a getServices packet to user1 by typing in the XML message directly into the chat window.



```
(9:46 AM) user1: <?xml version="1.0"
?><Body><MessageType>Communication</MessageType><Type>Roster</Type><item
jid="user2@129.241.208.209" name="user2"
subscription="both"><group>WEB</group></item></Body>
```

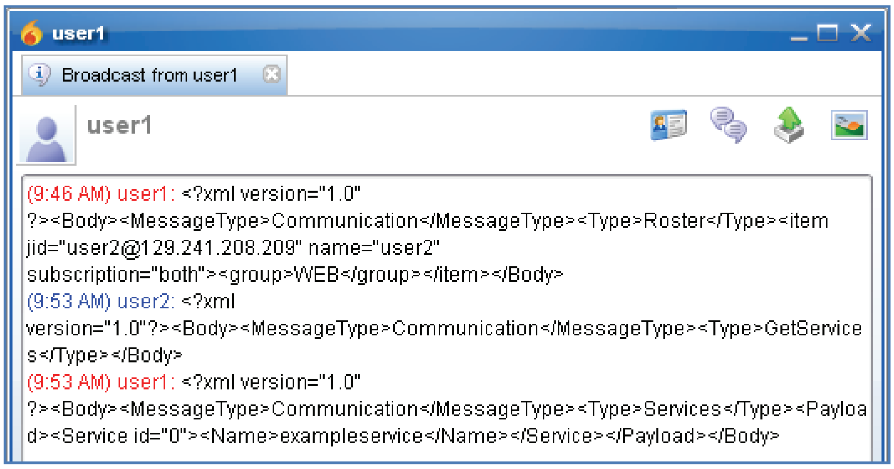
Figure 86 - User2 gets a rosterpacket from user1 when logging on



```
<?xml
version="1.0"?><Body><MessageType>Communication</MessageType>
<Type>GetServices</Type></Body>
```

Figure 87 - Typing an XML packet into the chat window

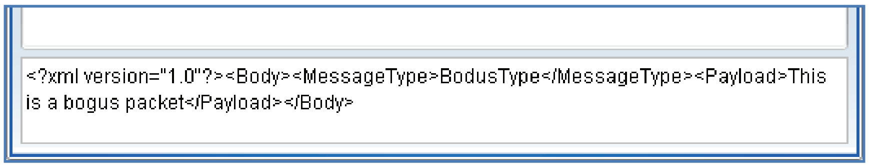
Pressing enter should give a feedback from user1 where he identifies his currently loaded services:



```
(9:46 AM) user1: <?xml version="1.0"
?><Body><MessageType>Communication</MessageType><Type>Roster</Type><item
jid="user2@129.241.208.209" name="user2"
subscription="both"><group>WEB</group></item></Body>
(9:53 AM) user2: <?xml
version="1.0"?><Body><MessageType>Communication</MessageType><Type>GetService
s</Type></Body>
(9:53 AM) user1: <?xml version="1.0"
?><Body><MessageType>Communication</MessageType><Type>Services</Type><Payloa
d><Service id="0"><Name>exampleservice</Name></Service></Payload></Body>
```

Figure 88 - GetServices yields services feedback

Now that the communication is tested with the standard functionality of the hscore plug-in, let's check how our new exampleservice works. Currently the only feedback programmed into the plug-in is to the eclipse console. This will trigger once it receives a message which does not have a <MessageType>Communication</MessageType> Notice that because no filters were specified in the trigger method made earlier, all packets except those destined for the communication module is received.



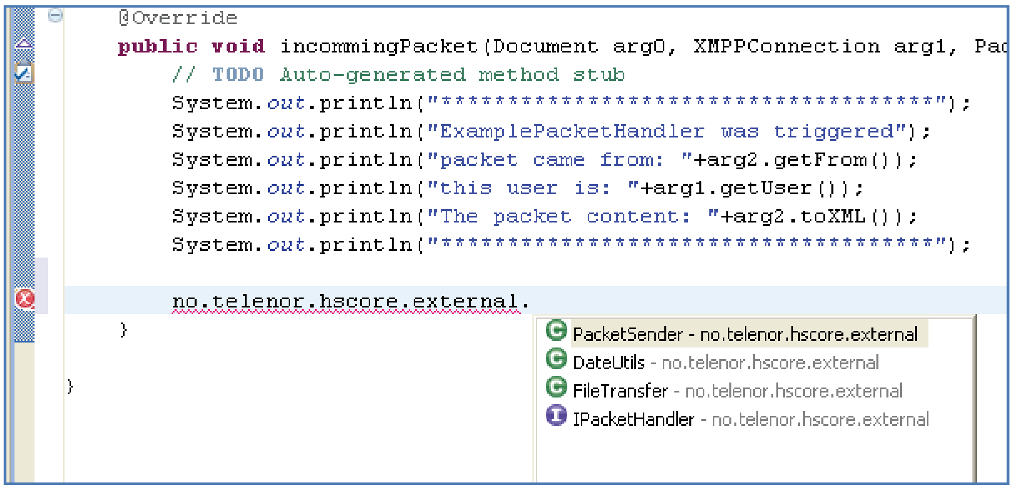
```
<?xml version="1.0"?><Body><MessageType>BodusType</MessageType><Payload>This
is a bogus packet</Payload></Body>
```

Figure 89 - Example creation of xml packet which triggers the new plug-in

```
******Connect's packetlistener has been triggered
MessageXML:      <message id="WbYbe-24" to="user1@129.241.208.209/MRL"
from="user2@129.241.208.209/spark" type="chat"><body>&lt;?xml
version=&quot;1.0&quot;?&gt;&lt;Body&gt;&lt;MessageType&gt;BodusType&lt;/MessageTyp
e&gt;&lt;Payload&gt;This is a bogus
packet&lt;/Payload&gt;&lt;/Body&gt;</body><thread>a2TEBU</thread><x
xmlns="jabber:x:event"><offline/><composing/></x></message>
Message Payload: <?xml
version="1.0"?><Body><MessageType>BodusType</MessageType><Payload>This is a bogus
packet</Payload></Body>
+++
MessageType: BodusType
*************************************
ExamplePacketHandler was triggered
packet came from: user2@129.241.208.209/spark
this user is: user1@129.241.208.209/MRL
The packet content: <message id="WbYbe-24" to="user1@129.241.208.209/MRL"
from="user2@129.241.208.209/spark" type="chat"><body>&lt;?xml
version=&quot;1.0&quot;?&gt;&lt;Body&gt;&lt;MessageType&gt;BodusType&lt;/MessageTyp
e&gt;&lt;Payload&gt;This is a bogus
packet&lt;/Payload&gt;&lt;/Body&gt;</body><thread>a2TEBU</thread><x
xmlns="jabber:x:event"><offline/><composing/></x></message>
*************************************
```

The first part of this textbox is printed by the hscore plug-in once it receives the message packet, the content is printed as well as the MessageType, which here is BodusType, clearly a bogus packet. After the dotted line in the middle of the text, the ExamplePacketHandler has been triggered and the print lines which were made earlier in this tutorial can be seen. This should be proof of concept regarding plugging new modules into the application.

Having the communication up and running, stop the application from eclipse and let’s send something back to the client. This is really easy at this point. Go back into the ExamplePacketHandler java class that was made earlier in this tutorial. All the useful utilities inside the hscore plug-in has been arranged in a separate package called external. Inside the incommingPacket method, type in “no.telenor.hscore.external.” and press Ctrl + spacebar for content assist. This will show the PacketSender, DateUtils and FileTransfer classes. These are utility classes which all plug-ins might use.

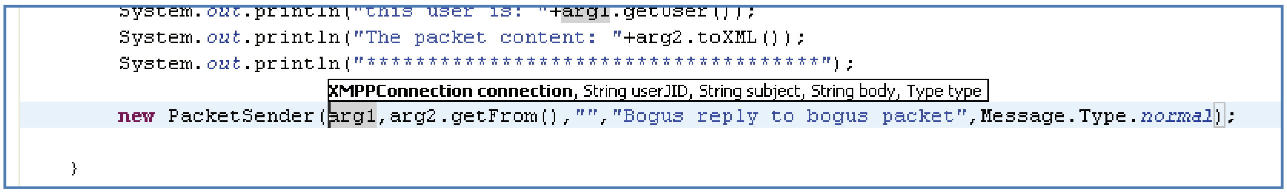


```
@Override
public void incommingPacket(Document arg0, XMPPConnection arg1, Pac
    // TODO Auto-generated method stub
    System.out.println("*************************************");
    System.out.println("ExamplePacketHandler was triggered");
    System.out.println("packet came from: "+arg2.getFrom());
    System.out.println("this user is: "+arg1.getUser());
    System.out.println("The packet content: "+arg2.toXML());
    System.out.println("*************************************");

    no.telenor.hscore.external.
}
}
```

PacketSender - no.telenor.hscore.external
DateUtils - no.telenor.hscore.external
FileTransfer - no.telenor.hscore.external
IPacketHandler - no.telenor.hscore.external

Figure 90 - no.telenor.hscore.external. available methods

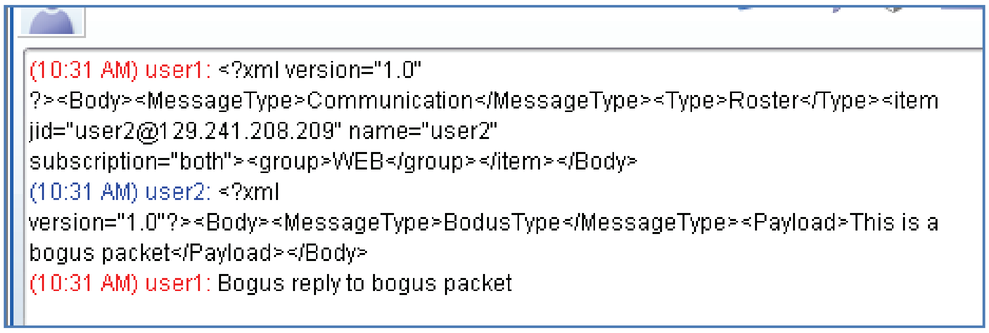
Use the PacketSender to send a packet back to the sender of the original packet.



```
        System.out.println("this user is: "+arg1.getUser());
        System.out.println("The packet content: "+arg2.toXML());
        System.out.println("*************************************");
                          XMPPConnection connection, String userJID, String subject, String body, Type type
        new PacketSender(arg1,arg2.getFrom(),"","Bogus reply to bogus packet",Message.Type.normal);

    }
```

Figure 91 - Sending a bogus packet to the packet sender

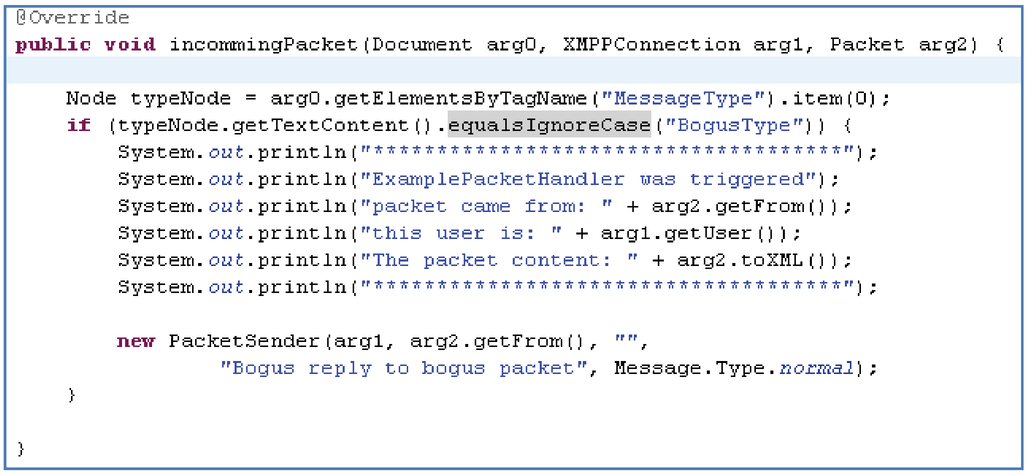
Notice how the PacketSender requires the XMPPConnection object which is received together with the incoming packet. The XMPPConnection is also quite useful for use in the new plug-in itself. It contains all the information necessary to use the connection with the XMPP server and the other clients in the roster. Next let's run the setup with the new PacketSender and see what we get.



```
(10:31 AM) user1: <?xml version="1.0"
?><Body><MessageType>Communication</MessageType><Type>Roster</Type><item
jid="user2@129.241.208.209" name="user2"
subscription="both"><group>WEB</group></item></Body>
(10:31 AM) user2: <?xml
version="1.0"?><Body><MessageType>BodusType</MessageType><Payload>This is a
bogus packet</Payload></Body>
(10:31 AM) user1: Bogus reply to bogus packet
```

Figure 92 - Replying to a packet with a simple message

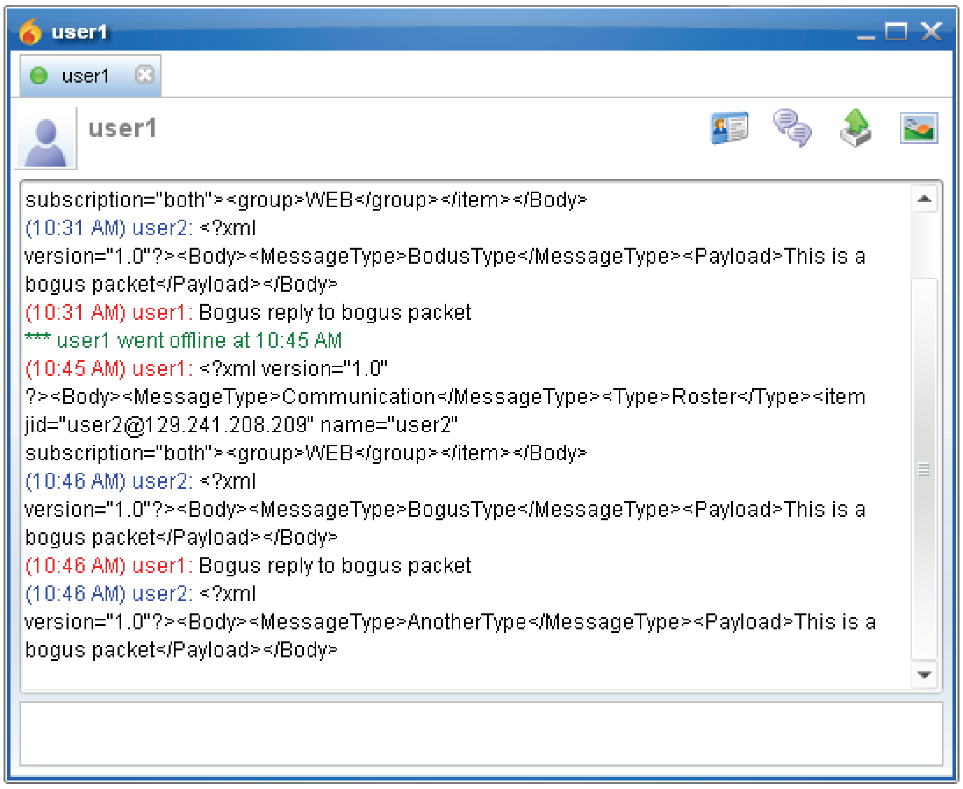
So far this tutorial has shown how to receive and send back a packet, but so far, no matter what kind of packet is received, the plug-in sends a bogus reply back, this is not a very powerful behavior. Because the incoming packets are parsed as XML, it is quite possible to read out tags from the packets and interpret them. This is what the hscore plug-in does when deciding which packet to forward to the different services.



```
@Override
public void incommingPacket(Document arg0, XMPPConnection arg1, Packet arg2) {

    Node typeNode = arg0.getElementsByTagName("MessageType").item(0);
    if (typeNode.getTextContent().equalsIgnoreCase("BogusType")) {
        System.out.println("*************************************");
        System.out.println("ExamplePacketHandler was triggered");
        System.out.println("packet came from: " + arg2.getFrom());
        System.out.println("this user is: " + arg1.getUser());
        System.out.println("The packet content: " + arg2.toXML());
        System.out.println("*************************************");

        new PacketSender(arg1, arg2.getFrom(), "",
                "Bogus reply to bogus packet", Message.Type.normal);
    }

}
```

Figure 93 - Adding a filter to the incoming packet method

After running the application again, try with the MessageType BogusType, then try to type in a messageType AnotherType, the last one will not trigger the bogus reply



```
subscription="both"><group>WEB</group></item></Body>
(10:31 AM) user2: <?xml
version="1.0"?><Body><MessageType>BodusType</MessageType><Payload>This is a
bogus packet</Payload></Body>
(10:31 AM) user1: Bogus reply to bogus packet
*** user1 went offline at 10:45 AM
(10:45 AM) user1: <?xml version="1.0"
?><Body><MessageType>Communication</MessageType><Type>Roster</Type><item
jid="user2@129.241.208.209" name="user2"
subscription="both"><group>WEB</group></item></Body>
(10:46 AM) user2: <?xml
version="1.0"?><Body><MessageType>BogusType</MessageType><Payload>This is a
bogus packet</Payload></Body>
(10:46 AM) user1: Bogus reply to bogus packet
(10:46 AM) user2: <?xml
version="1.0"?><Body><MessageType>AnotherType</MessageType><Payload>This is a
bogus packet</Payload></Body>
```

Figure 94 - Testing the BogusType filter - AnotherType does not yield a reply.

6.1.1.6 *Exporting the plug-in*

This tutorial has shown how to create a plug-in for the hscore application and test it inside the comfort of Eclipse, now let's try and deploy the new plug-in and test it in the readymade hscore application. During this stage, debugging becomes much more of a hassle compared to what it was like inside Eclipse. This is largely because of the unpolished state of the hscore plug-in, adding more logging to the hscore plug-in is a prioritized task for the next release.

The first step is to go to the overview tab in the plugin.xml file in the new exampleservice project. To the bottom right there is an Export area. Click on number four, the Export Wizard. In the pop-up, choose a destination for your new plug-in. Right now, choose a directory since we are going to use it right away to test the plug-in functionality. Later on, it is easier to distribute the plug-in in as an archive. Make sure the exampleservice checkbox at the top is checked. Click finish. Browse to the destination directory.

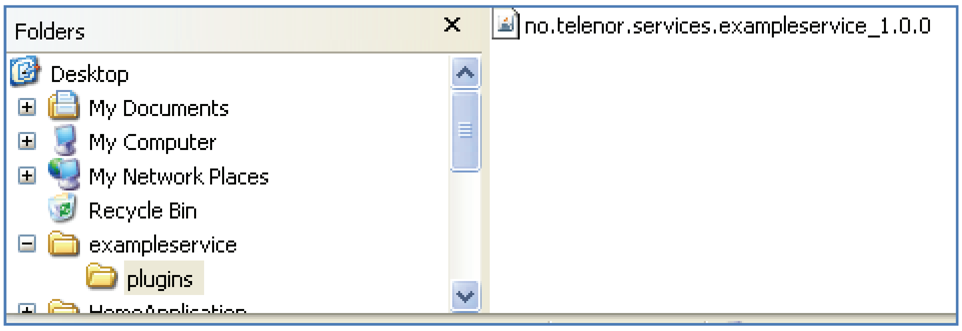
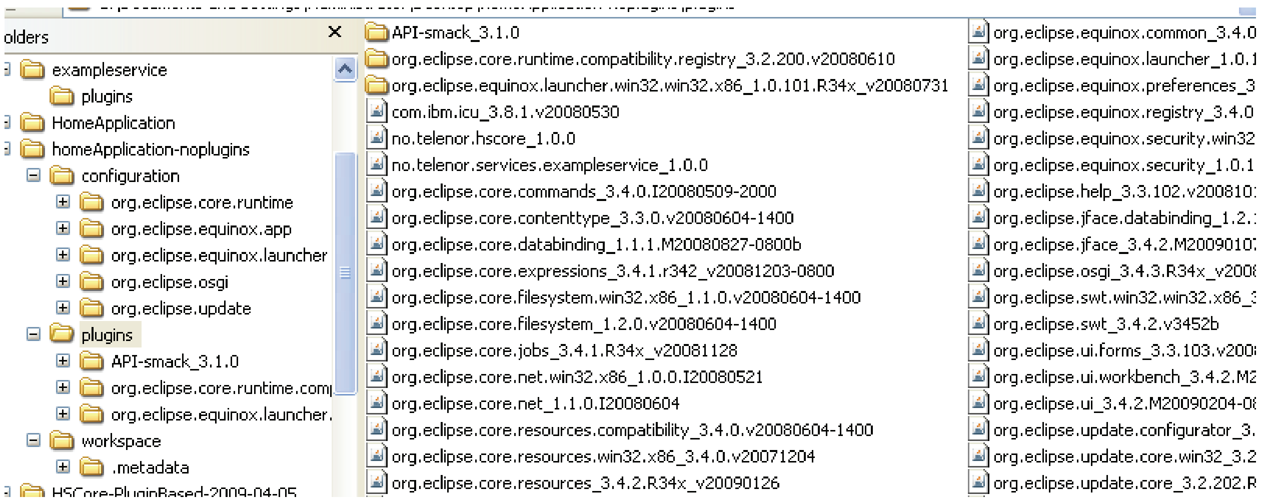


Figure 95 - The newly exported plug-in

The new plug-in is simply a jar file, in a plugins folder. Copy the plugins folder to the clipboard and browse to the folder of the home application without plug-ins which was referenced in the beginning of this tutorial. Notice how this application also has a plugins folder. Paste the plugins folder into the folder of the home application. Click "Yes" when asked to overwrite the plug-ins folder. Since the two folders do not contain the same contents, they will be merged.

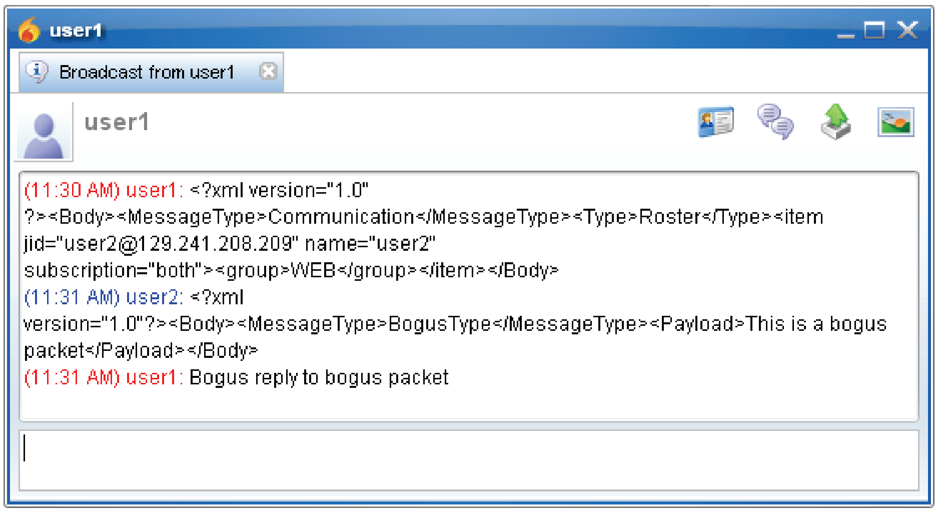

```
olders
exampleservice
    plugins
HomeApplication
homeApplication-noplugins
    configuration
        org.eclipse.core.runtime
        org.eclipse.equinox.app
        org.eclipse.equinox.launcher
        org.eclipse.osgi
        org.eclipse.update
    plugins
        API-smack_3.1.0
        org.eclipse.core.runtime.com
        org.eclipse.equinox.launcher.
    workspace
        .metadata
HSCore-PluginBased-2009-04-05
```

```
API-smack_3.1.0
org.eclipse.core.runtime.compatibility.registry_3.2.200.v20080610
org.eclipse.equinox.launcher.win32.win32.x86_1.0.101.R34x_v20080731
com.ibm.icu_3.8.1.v20080530
no.telenor.hscore_1.0.0
no.telenor.services.exampleservice_1.0.0
org.eclipse.core.commands_3.4.0.I20080509-2000
org.eclipse.core.contenttype_3.3.0.v20080604-1400
org.eclipse.core.databinding_1.1.1.M20080827-0800b
org.eclipse.core.expressions_3.4.1.r342_v20081203-0800
org.eclipse.core.filesystem.win32.x86_1.1.0.v20080604-1400
org.eclipse.core.filesystem_1.2.0.v20080604-1400
org.eclipse.core.jobs_3.4.1.R34x_v20081128
org.eclipse.core.net.win32.x86_1.0.0.I20080521
org.eclipse.core.net_1.1.0.I20080604
org.eclipse.core.resources.compatibility_3.4.0.v20080604-1400
org.eclipse.core.resources.win32.x86_3.4.0.v20071204
org.eclipse.core.resources_3.4.2.R34x_v20090126
```

```
org.eclipse.equinox.common_3.4.0
org.eclipse.equinox.launcher_1.0.1
org.eclipse.equinox.preferences_3
org.eclipse.equinox.registry_3.4.0
org.eclipse.equinox.security.win32
org.eclipse.equinox.security_1.0.1
org.eclipse.help_3.3.102.v200810
org.eclipse.jface.databinding_1.2.
org.eclipse.jface_3.4.2.M2009010
org.eclipse.osgi_3.4.3.R34x_v200
org.eclipse.swt.win32.win32.x86_
org.eclipse.swt_3.4.2.v3452b
org.eclipse.ui.forms_3.3.103.v200
org.eclipse.ui.workbench_3.4.2.M2
org.eclipse.ui_3.4.2.M20090204-0
org.eclipse.update.configurator_3.
org.eclipse.update.core.win32_3.2
org.eclipse.update.core_3.2.202.R
```

Figure 96 - Added the exampleservice to the plugins folder of the homeapplication

Start the home application by executing the homeapplication.exe file. Since the plug-in does not yet contain a GUI component, it does not show up in the application. To be able to test that the plug-in was successfully loaded into the application, use the same procedure as when testing inside Eclipse. Log the application in as user1, log the spark client in as user2 and send the BogusType message to user 1 as shown in Figure 97. The feedback should be the same, "Bogus reply to bogus packet"



```
(11:30 AM) user1: <?xml version="1.0"
?><Body><MessageType>Communication</MessageType><Type>Roster</Type><item
jid="user2@129.241.208.209" name="user2"
subscription="both"><group>WEB</group></item></Body>
(11:31 AM) user2: <?xml
version="1.0"?><Body><MessageType>BogusType</MessageType><Payload>This is a bogus
packet</Payload></Body>
(11:31 AM) user1: Bogus reply to bogus packet
```

Figure 97 - Testing the service outside Eclipse

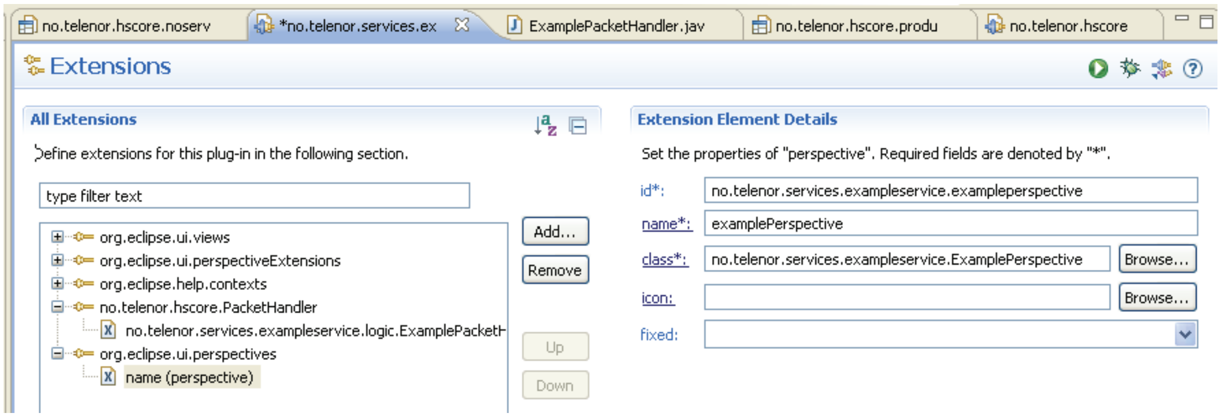
So far, this tutorial has shown how a plug-in can be developed and tested inside Eclipse, exported to the hard drive, copied into the plugins folder of the home application and executed as a part of the home application. Deploying the new service is a matter of distributing the jar to the users and telling them to add it to their plugins folder. Eclipse RCP contains a new update function called p2, which could be included in a future release of the home application; this might make it even simpler to distribute updates as this opens up the possibility of self-updating, automatic updating and more from inside the home application.

6.1.1.7 Contributing to the GUI

At this point it is quite easy to make contributions to the GUI of the application. The hardest part is to actually create a compelling and smooth GUI. Eclipse uses SWT, the standard Widget toolkit to create its GUI components. It would be a good idea to study this area before trying to create compelling graphics for the new plug-in.

In Eclipse, each application has an arbitrary number of Perspectives; each of these perspectives contains any number of Views and Editors, organized in a customizable fashion and with the option of being re-organized by the user. These Perspectives are found dynamically and show up in the perspectives menu in the top left corner, because all perspectives extend the perspective extension point. The following will explain how to get a new perspective into the home application. It will not go into any detail concerning the GUI itself as this is well documented in the many SWT tutorials and books on the subject [www.eclipse.org/swt]

Go to the extensions tab of the plugin.xml file. Notice how a view is already present. This was created because a template was used when first creating the plug-in. Click Add… select the org.eclipse.ui.perspectives extension point and Finish.



Define extensions for this plug-in in the following section.

type filter text

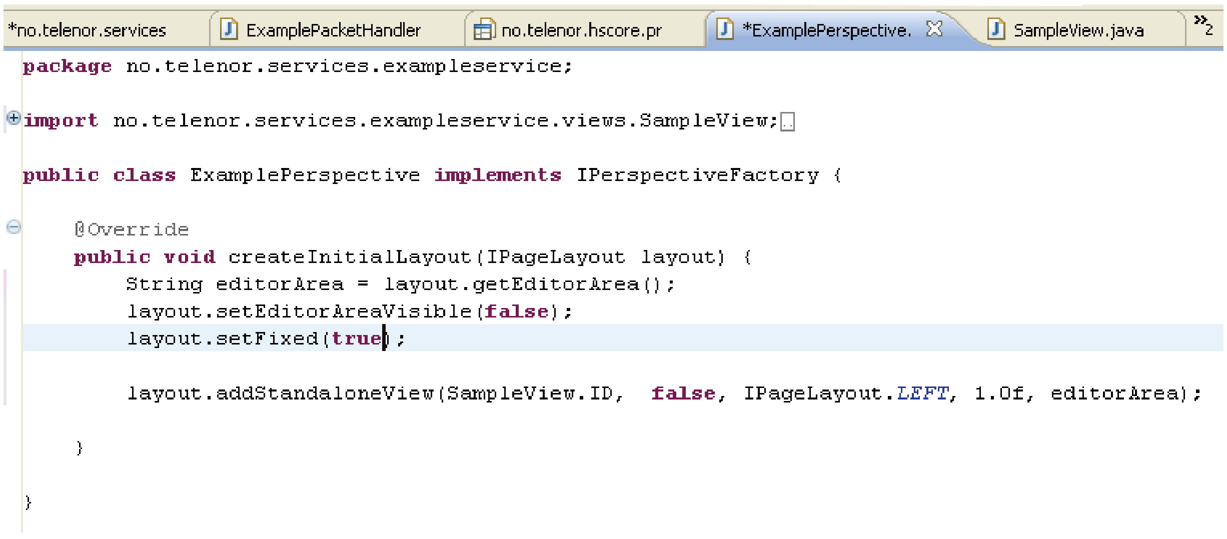
- org.eclipse.ui.views
- org.eclipse.ui.perspectiveExtensions
- org.eclipse.help.contexts
- no.telenor.hscore.PacketHandler
  - no.telenor.services.exampleservice.logic.ExamplePacketH
- org.eclipse.ui.perspectives
  - name (perspective)

Set the properties of "perspective". Required fields are denoted by "*".

id*:	no.telenor.services.exampleservice.exampleperspective
name*:	examplePerspective
class*:	no.telenor.services.exampleservice.ExamplePerspective
icon:	
fixed:	

Figure 98 - Adding a perspective to the new plug-in

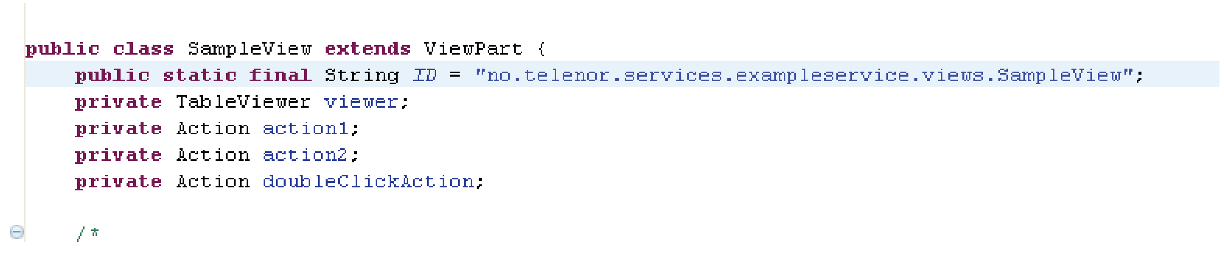
Alter the name and ID of the new perspective, Clicking on the “class*:” link will create a new perspective class with the current name. Add the following four lines to the createInitialLayout method:



```
package no.telenor.services.exampleservice;

import no.telenor.services.exampleservice.views.SampleView;

public class ExamplePerspective implements IPerspectiveFactory {

    @Override
    public void createInitialLayout(IPageLayout layout) {
        String editorArea = layout.getEditorArea();
        layout.setEditorAreaVisible(false);
        layout.setFixed(true);

        layout.addStandaloneView(SampleView.ID,  false, IPageLayout.LEFT, 1.0f, editorArea);

    }

}
```

Figure 99 - Adding a view to the Perspective

Set the Sample View's ID in the sampleView class which should already be present in the no.telenor.services.exampleservice.views package from the creation of the plug-in.



```
public class SampleView extends ViewPart {
    public static final String ID = "no.telenor.services.exampleservice.views.SampleView";
    private TableViewer viewer;
    private Action action1;
    private Action action2;
    private Action doubleClickAction;

    /*
```

Figure 100 - Explicitly setting the SampleView's ID

Save the project, and run the hscore product as shown earlier in this tutorial.

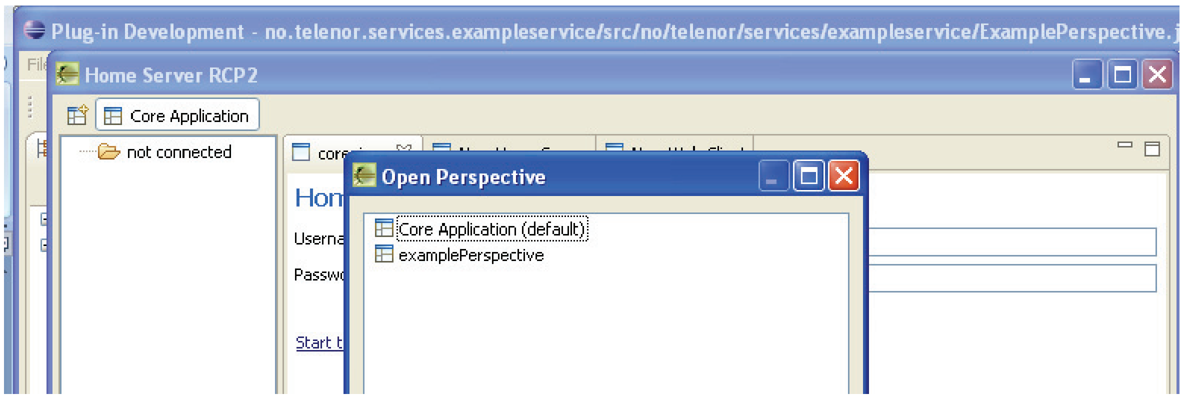

Plug-in Development - no.telenor.services.exampleservice/src/no/telenor/services/exampleservice/ExamplePerspective.j
Home Server RCP2
Core Application
not connected
Open Perspective
Core Application (default)
examplePerspective
Hon
Userna
Passw
Start t

Figure 101 - The perspectives in the home server application after clicking on the top left button

It is now possible so click the top-left perspectives button in the application and get the new example perspective as an option. Clicking it will give the GUI in Figure 102. The examplePerspective now only shows an example GUI from the template created early on in this tutorial and leaves any fancy GUI up to the plug-in developer.

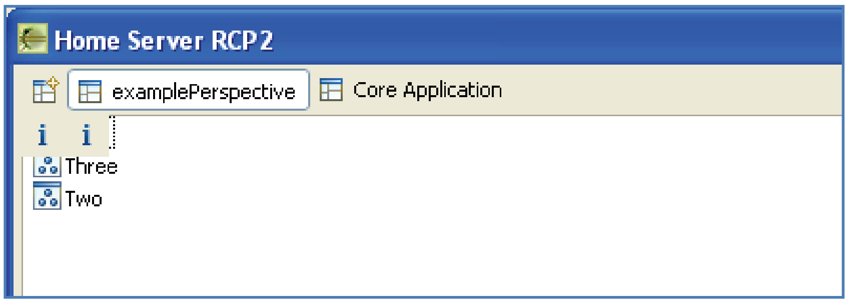


Figure 102 - The examplePerspective

6.1.1.8 Concluding the tutorial

This tutorial has been a basic walkthrough of a typical plug-in development scenario in an Eclipse RCP application. The scenario is very similar to an Eclipse plug-in scenario, the major difference being the custom extension points this plug-in is connecting to. Being the author of the hscore plug-in, it is possible to create and customize both sides of the application to adhere to custom specifications.

6.2 Appendix B – Code and applications

6.2.1 Home Application

The home application has been described in the design and implementation of the thesis. This is a description of the attached code. The code will not be included in this document but will rather be included as an attachment. This decision was made because of the large amount of code. All code concerning the Home application is in Java, designed to run in the Eclipse RCP framework. It must be compiled in the Eclipse RCP framework, otherwise it will not run. The following environment has been used to develop the code [151].

- Eclipse-rcp-ganymede-sr2, version 3.4.2
    - Eclipse delta pack has been used to compile the code for different platforms
    - It might be necessary to use eclipse 3.4.x to compile the code out of the box, since eclipse changes some in each release. After some modification it should also compile in eclipse 3.5 which will be released during the summer of 2009
- Sun Java SE 6, it should also work with Sun java SE 5. This can be set in the product.xml file
- Windows XP 32 bit and Vista 64 bit OS has been used in the development, but should not have any impact.

The code will be included in several setups, for simplicity and readability.

- The latest version of all the java projects has been included in an eclipse workspace file structure. This should be used for code review and getting to know the code.
- For historic purposes, a dump of the SVN repository has been included. This repository includes all the commits to the project, from the start of the coding. The current projects are located in the root directory, while old and deprecated projects are located in a separate folder in the repository. The repository dump can be accessed by following the instructions for moving a SVN repository found here [http://b.lesseverything.com/2007/3/12/moving-svn-from-one-machine-to-another] If the development is continued it will make sense to use this repository, or create a new one from the latest version of the code included above.

Recommended tutorials and books:

- www.vogella.de / eclipse RCP tutorials
- Developing Eclipse RCP applications, book from eclipse foundation books (will be a new edition in September
- Developing professional grade plug-ins for the eclipse platform

6.2.2 Network Server

The network server contains all the code for the web site, mostly JavaScript. In addition to this, the configuration file for the apache2 server, the httpd.conf file is included. With this file it should be easy to set up an apache2 server on any Linux box, copy the httpd.conf file to the /etc/apache2/ directory and the apache server should have the right behavior for the system. The most notable thing is the forwarding proxy of all http-bind requests to the Openfire server.

The Debian package for the Openfire XMPP server has also been included. The documentation for installing and configuring this can be found at the Jive Software website. The standard configuration is used. For scalability issues, the external database should be used. For this thesis, the MySQL database in Ubuntu has been used. Just set up a standard database in Ubuntu first, then when doing the installation of Openfire, give the database parameters and it will fix the rest.

The network server has consisted of a computer with Ubuntu server 8.10, with Ubuntu-desktop-gnome window manager. The GUI has been useful because the network server has been running the NSclient, and this has been made with a GUI from Eclipse RCP.

The NS client has been included in the SVN repository which can be found in the home Application directory.