Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Master's thesis

Liferay communication infrastructure

Bc. Marcel Mika

Supervisor: Ing. Pavel Kordík, Ph.D.

10th May 2013

Acknowledgements

I would like to thank and express my appreciation to Ing. Pavel Kordík, Ph.D. for his invaluable contribution throughout the development of this project, as well as the Czech Technical University in Prague for one of the most inspiring periods of my life. Also, a special thanks to Angela Scarselli for her proofreading help. Last, but certainly not least, I would like to thank my family and Bc. Miroslava Horáková for their amazing support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60(1) of the Act.

In Prague on 10th May 2013 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

MIKA, Marcel. *Liferay communication infrastructure.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2013.

Abstract

The thesis investigates a modern instant messaging communication system in the Liferay portal environment. Such software is currently a widely used form of direct, real-time communication. The given system consists of three plugins. The first plugin, imported to the Liferay portal, serves as a user interface and a gate to the Openfire server. It is divided into the backend and frontend side. The backend side is responsible for the communication with the Openfire server and also for the creation of view. The frontend side handles all user actions via the modern, interactive javascript interface and communicates with backend via AJAX calls. The second plugin, imported to the Openfire server, creates public chat rooms based on the data from the KOSapi service. The third plugin, imported to the Openfire server, serves as a platform for the social network analysis which will visualize connections between participants and thus show interesting social patterns that can be used for a variety of different purposes.

The final product will be a part of the emerging University Information System which is currently in the phase of development. It aims to be a frequently used channel that will connect multiple groups like students, teachers, employees and alumni. It should also be ready for the Liferay marketplace and thus distributed among a large group of Liferay portal instances.

Keywords Liferay, Openfire, XMPP, Instant Messaging System, Social Network Analysis

Abstrakt

Následující text popisuje analýzu, návrh, implementaci a nasazení ,,instant messaging" komunikačního systému v prostředí portálu Liferay, jenž byl vyvinut v rámci této diplomové práce. Tento druh software je momentálně široce používaným způsobem přímé komunikace v reálném čase. Systém se skládá ze tří částí ve formě tzv. pluginů. První plugin, jenž lze importovat do Liferay, slouží jako uživatelské rozhraní a také jako klient serveru Openfire. Dělí se na dvě části: backend, který je zodpovědný za komunikaci s Openfire serverem a také za vytvoření zobrazovací části a frontend, který zpracovává všechny uživatelovy podněty skrze moderní a interaktivní javascriptové rozraní. S backendem komunikuje pomocí volání AJAX. Druhý plugin importovaný do serveru Openfire vytváří veřejné místnosti na základě dat ze služby KOSapi. Poslední plugin pak slouží jako platforma pro analýzu sociálních sítí. Jejím cílem je vizualizace spojení mezi uživateli umožňující zobrazovat podnětné vzory chování, které mohou být využity pro rozmanité účely.

Finální produkt bude součástí nově vznikajícího univerzitního informačního systému. Cílem celé práce je vytvořit intenzivně využívaný informační kanál, s jehož pomocí bude možné spojit různorodé skupiny uživatelů, kterými jsou například studenti, kantoři, zaměstnanci školy či absolventi. Finální produkt je připraven k distribuci pomocí Liferay marketplace, čímž se zaručuje jeho využití v rámci velkého množství Liferay instancí.

Klíčová slova Liferay, Openfire, XMPP, Instant Messaging System, Analýza sociálních sítí

Contents

List of Figures

Introduction

Motivation

I have always been fascinated by the growth of modern communication platforms. I still remember the video of Mark Zuckerberg talking about a new way of communication that will make e-mail history. One of the reasons I chose this project for my thesis was that Liferay, such a widely used platform, does not have a decent tool for direct instant messaging communication. The recent implementation of Liferay Chat Portlet just does not fulfill my high requirements. Given these facts, I realized that there might be a small gap in the market.

The solution I am about to implement will be a part of the University Information System. This provides a great testing opportunity with many users who can potentially generate a lot of feedback.

Furthermore, the thesis includes a chapter about social network analysis, which might be considered the key research technique in modern sociology. I really appreciate such trends that have emerged in the last couple years. The computer is now more social and human than ever before. As a result, machines can decide or deliver content based on user behavior. In the scope of this project, social network analysis may optimize decisions during the student schedule composition process and may show trending topics within the community.

Thesis goals

The main focus of this project is to create a modern instant messaging communication system in the environment of a Liferay [36] portal. As aforementioned, such software is currently a widely used form of direct, real-time communication. It would appear on nearly every page within the portal, which makes it even more interactive. This would provide the

possibility to send text messages not only between two individuals, but also across groups.

The final product will be a part of the emerging University Information System, which is currently in the development phase. It aims to be a frequently used channel that will connect multiple groups like students, teachers, employees and alumni. It should be also ready for the Liferay marketplace and thus distributed among a large group of Liferay portal instances.

Last but not least, all conversations will be stored on a separate secured server. Therefore, it would be possible to perform a social network analysis on data located in the server database. The thesis also aims to create a platform that will visualize connections between participants and thus show interesting social patterns that can be used for a variety of different purposes.

About the text

The following text is divided into chapters based on the standard software development process. The *State of the Art* chapter discusses existing available implementations of similar software and technology. The *Analysis and Design* chapter is introduced by a list of functional requirements and subsequently covers the software architecture in a broader view. It describes all components within the system and the way they communicate, the relationships among them, and their behaviour and properties. The Design section examines the more in-depth specification of each component. Properties and methods are described with a set of UML[1] diagrams. The *Implementation* chapter discusses the implementation process. The *Testing and Examples* chapter describes testing methodologies and results. The *Conclusion* serves as a summary of the thesis results and contributions. It also includes a future improvement section, which lists all possible improvements that have not yet been implemented.

[1] UML – Unified Modeling Language [49]

Chapter 1

State of the Art

This chapter sheds insight into the current level of development in the field of instant messaging. Facebook[1] and Google+[2] were chosen as references because they are both currently widely used services and therefore mirror the latest evolution. A discussion of social network analysis and an example of the graph visualization software – Gephi [8] will follow. Then the XMPP[3] protocol is introduced, as protocol is a leading standard for presence and real-time messaging [23]. The adoption of XMPP protocol is a must for any instance messaging service that tends to communicate with the outside world. Subsequently, Openfire, which was chosen as a XMPP server for this project, will be described. Liferay portal, as a leading platform, and Liferay Chat Portlet, as a current implementation of XMPP client, are described at the end of this chapter.

1.1 Instant Messaging

Nowadays, communication via instant messaging is widely used. Thanks to this kind of software, users can get a response a short time after sending the message. For the majority of people, it is cheaper than phone calls and faster than e-mail. Moreover, the newest IM software has integrated voice chatting, data transmission, video conferencing and other advanced features.

[1] Facebook is a social utility that connects people with friends and others who work, study and live around them. [4]

[2] Google+ is a social network from Google [14]

[3] XMPP – Extensible Messaging and Presence Protocol [56]

1.1.1 History

As described in [32, p. 201], instant messaging use expanded with the introduction of the ICQ[4] in November 1996. Since then AOL[5], Microsoft MSN Messenger, Yahoo! Messenger, and others were introduced and adopted by the public. Nowadays, instant messaging has become integrated into other services rather than being independent. Particularly, Facebook and Google have instant messaging built into their user interfaces.

1.1.2 Facebook Chat

Facebook's instant messaging feature is integrated into their user interface but can also be accessed via XMPP protocol [5]. Consequently, users may use their own clients as long as they implement given protocol.

Chat has two user interfaces. Firstly, users can chat via small panels related to each conversation 1.1. The panels are visible on each page hence easily accessible and interactive. The second option is to use a separate chat page, which is more comfortable due to the larger visible area, but less interactive.

Conversations can be one-to-one or many-to-many. They can be created by either clicking on the user name in the list of friends or via the "Send a New Message" option in the list of conversations. Any participant who is already in the conversation can invite other participants via the "Add more friends to chat" button on the conversation panel.

Whenever the user receives incoming message he or she is notificated by the following:

- Page title is extended with the number of unread messages[6].
- Conversation icon and conversation panel are extended with the badge which contains number of unread messages.
- Audio notification is played[7].

1.1.3 Google+ Chat

Google+ [14] offers features similar to Facebook. On the other hand, one of the key differences is that the chat panels are used only for one-to-one

[4] I Seek You – Instant messaging computer program
[5] AOL – America Online is a mass media corporation from New York
[6] e.g. for five unread messages: (5) Facebook
[7] User can always turn this option off.

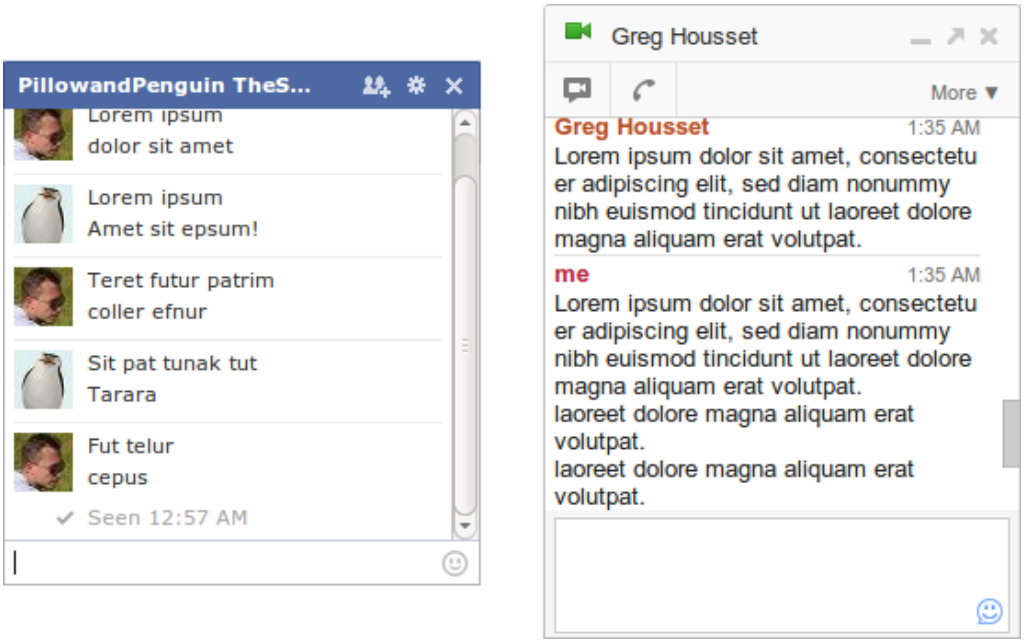

Figure 1.1: Facebook Chat panel vs. Google+ Chat panel

communication[8]. If the user wants to interact with more friends, Google Hangouts needs to be started.

Thanks to Google Hangouts [16], users may videochat with up to nine friends, share screens, use Google Drive and other Google technologies. Such integration brings communication to a completely new level.

1.2 Social Network

People connect with others through social networks formed by kinship, language, trade, exchange, conflict, citation and collaboration [19, p. 31]. In the scope of this project, connections are made by sending messages within private conversations. Figure 1.2 shows a simple social network. It comprises four nodes (A,B,C,D) which represent four individuals and arrows which represent an interpersonal interaction between them (linkage). While individual D does not interact with A and C, he/she is part of the network through his/her relationship with B.

[8] Although the many-to-many chat is available in Google Talk [15] it is not available in the Google+ Chat. Google Talk represents similar service based on the XMPP protocol like e.g. Facebook Chat.

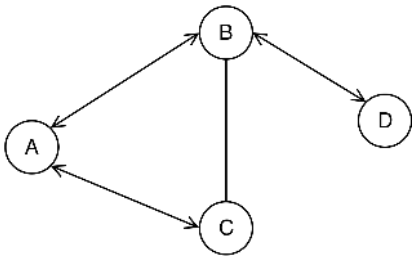

Figure 1.2: Example of a simple social network

Social network theory is described as a: "set of theories for forecasting, reasoning about, and understanding how social networks form, are maintained, and evolve, and the role of variables such as social networking tools, media, and stress in affecting the emergence, utilization, management, and change in social networks." [48].

Social network analysis is defined as a: "tool that can be used to analyze the structure of interpersonal relationships within a group of individuals. These relationships, taken collectively, constitute a network. SNA treats individuals as nodes and the relationship between individuals as linkages" [51, p. 49]

Network analysis metrics allow analysts to examine social networks. Initially, they focused on simple counts of connections. However, their studies evolved after the inventions of concepts like density, centrality, structural holes, balance, and transitivity. Given concepts are described here [19].

A graphical representation of a social network can be visualized and explored via the social network tools (e.g. Gephi).

1.3 Gephi

Gephi [9] is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs [8]. It facilitates the creation of social data connectors in order to map community organizations and small-world networks. The statistics and metrics framework offer the most common metrics for social network analysis:

- Betweenness,
- Closeness,
- Diameter,
- Clustering Coefficient,

- Average shortest path,
- Community detection (Modularity).

Gephi accepts multiple graph formats (e.g. GEXF, GDF, GML, GraphML, CSV, Spreadsheet, and others) [10]. The GEXF[9] file was chosen as a main format for this project.

Overview

Current stable version: 0.8.2 Beta
Platforms: Windows, Linux, MacOS X
Language: Java 1.6
License: CDDL + GNU GPL 3

1.4 Gexf4j

The GEXF file format is distributed as an XML file. Gexf4j [6][7] serves as a Java library, which can easily generate such files. This library is used to create and write GEXF files for visualizing graphs using Gephi and any other GEXF-supporting application. It supports GEXF 1.2 format.

Overview

Current stable version: 0.4.2-BETA
Language: Java
License: Apache License 2.0

1.5 XMPP Protocol

XMPP (formerly known as Jabber) is the leading open standard for presence and real-time messaging [23]. It has been an approved standard of the IETF[10] since 2004 and is maintained by the Jabber Software Foundation. Today, XMPP is used by leading companies and has millions of users worldwide. Although the core technology is stable, the XMPP community continues to define various XMPP extensions through an open standards process run by the XMPP Standards Foundation [56].

[9] GEXF – Graph Exchange XML Format [20]

[10] The Internet Engineering Task Force is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet [21].

1.6 Openfire

Liferay Chat Portlet (will be described later) directly supports the Openfire server [27] via the Smack API Library [25]. Since Openfire was already chosen by the Liferay Chat Portlet developers, there was no need to look for other Jabber servers.

Jive Software refers to Openfire as a: "real time collaboration (RTC) server licensed under the Open Source Apache License. It uses the only widely adopted open protocol for instant messaging, XMPP (also called Jabber)." [24] XMPP protocol is almost fully implemented [30]. Moreover, it supports all the features that are needed to satisfy the goals of this thesis:

- Plugin interface;
- SSL/TLS[11] support;
- Offline Messages support;
- Server-to-Server connectivity;
- Database connectivity for storing messages and user details (including the embedded HSQL database and support for MySQL, PosgreSQL and other databases);
- LDAP[12] integration;
- Platform independent (with the installers for different platforms);
- Connection manager for load balancing;
- Message archiving-logging;
- User-friendly web-based installation and administration panel;
- Multi-user chat.

Overview

Current stable version: 3.8.1
Programming language: Java
Licence type: Apache License 2.0

[11] SSL/TLS – Secure Sockets Layer and Transport Layer Security are cryptographic protocols used over the Internet.

[12] LDAP – Lightweight Directory Access Protocol is a protocol for accessing and maintaining distributed directory information services over the Internet.

1.7 Smack API

Smack [31][25] is an XMPP client library for instant messaging and presence. It is already embedded in the Liferay Chat Portlet. It provides full XMPP integrations such as sending notification messages and presence-enabling devices.

Overview

Current stable version: 3.2.2
Programming language: Java
Licence type: Apache License 2.0

1.8 Liferay

Liferay [38] serves as a portal. According to Sezov a portal might be described as: "A single web-based environment from which all of a user's applications can run. These applications are integrated together in a consistent and systematic way." [53] Basically, Liferay is a container for small applications that encapsulate functionality. These applications are called portlets.

The given approach promises a lot of flexibility and scalability since each portlet is an independent unit with strictly defined boundaries. It can be developed separately and easily integrated into the portal afterwards. Portlets should be platform independent. With their strictly defined boundaries and interfaces as described in the JSR-286 standard [50], it should be possible to integrate any portlet into any portal.

Liferay possess their own marketplace [39] similar to Google Play [13] and the Apple App Store [2]. Currently, the marketplace only accepts the submission of free applications. Paid licensing options will be added in the coming months.

There have been more than 4 million downloads of Liferay to date. Based on the product overview [42], it is also an independent market leader. The source code is public and under LGPL[13] licence, and it can thus be linked to software that is not open source. The source code is available on Github [11].

[13]LGPL – Lesser General Public License is a free software licence which allows developers to integrate software under the LGPL licence into their own without being required to release the source code.

Based on these points, Liferay was chosen as the leading platform for the emerging University Information System developed by the Faculty of Information Technology, Czech Technical University in Prague. Therefore, the selection of technologies should consider Liferay as a main platform for any further development.

1.8.1 Difference between a portlet and a plugin

To clarify further reading it is necessary to distinguish between a portlet and plugin:

- *portlet* - documentation defines portlets as: "pluggable user interface software components that are managed and displayed in a web portal. Portlets produce fragments of markup code that are aggregated into a portal page. Typically, following the desktop metaphor, a portal page is displayed as a collection of non-overlapping portlet windows, where each portlet window displays a portlet."[33];

- *plugin* - is defined as an: "umbrella term for installable portlet, theme, layout template, hook, Ext and webmodule Java EE .war files. Though Liferay comes bundled with a number of functional portlets, themes, layout templates, hooks and web modules, plugins provide a means of extending Liferay to be able to do almost anything."[41].

In other words, a portlet is an encapsulated functionality in the form of a window that can be dragged and dropped by the user to the area of the portal. Hence the user is usually the final authority on deciding which portlets are going to be displayed. The plugin, on the other hand, serves as an extension to the Liferay portal core functionality. Thus the plugin is most likely dealt with by the portal administrator.

Overview

Current stable version: 6.1
Programming language: Java
Licence type: LGPL

1.9 Alloy UI

Liferay uses Alloy UI [1][47] for the javascript part of its functionality. Alloy is a UI framework built on top of YUI3[14] that provides a simple API for building highly scalable applications. It is a JavaScript library, a CSS framework, a set of HTML recipes and a taglib library, all combined to empower developers across multi-skilled teams deliver rich and dynamic applications.

Overview

Current stable version: 2.0
Programming language: Javascript
Licence type: BSD

1.10 Liferay Chat Portlet

As mentioned in the previous chapter, one part of the solution will be in the form of a plugin to the Liferay portal. This makes the scope of possible existing implementations relatively narrow. Currently there is only one major implementation – Liferay Chat Portlet.

The user guide describes it as a component that "provides a convenient way of allowing users to send each other instant messages when they are logged into your web site. It appears as a bar at the bottom of every page, showing who is logged on, their statuses, and any chats the logged-in user has open."[44] The Portlet is also hosted as an open source project on Github [12].

Currently there is no specific documentation on the Liferay Chat Portlet. All necessary information was taken by reverse engineering and reading the code. In further text, all occurrences of Liferay Chat Portlet will be related to the existing portlet developed by Liferay. Any occurrence of Liferay Chat Plugin will be related to the newly created plugin based on the Liferay Chat Portlet.

The Portlet might be divided into 2 parts; the Frontend and Backend (Figure 1.3). They communicate with each other via AJAX.

[14]YUI – Yahoo User Interface is a free, open source JavaScript and CSS library for building richly interactive web applications. [58]

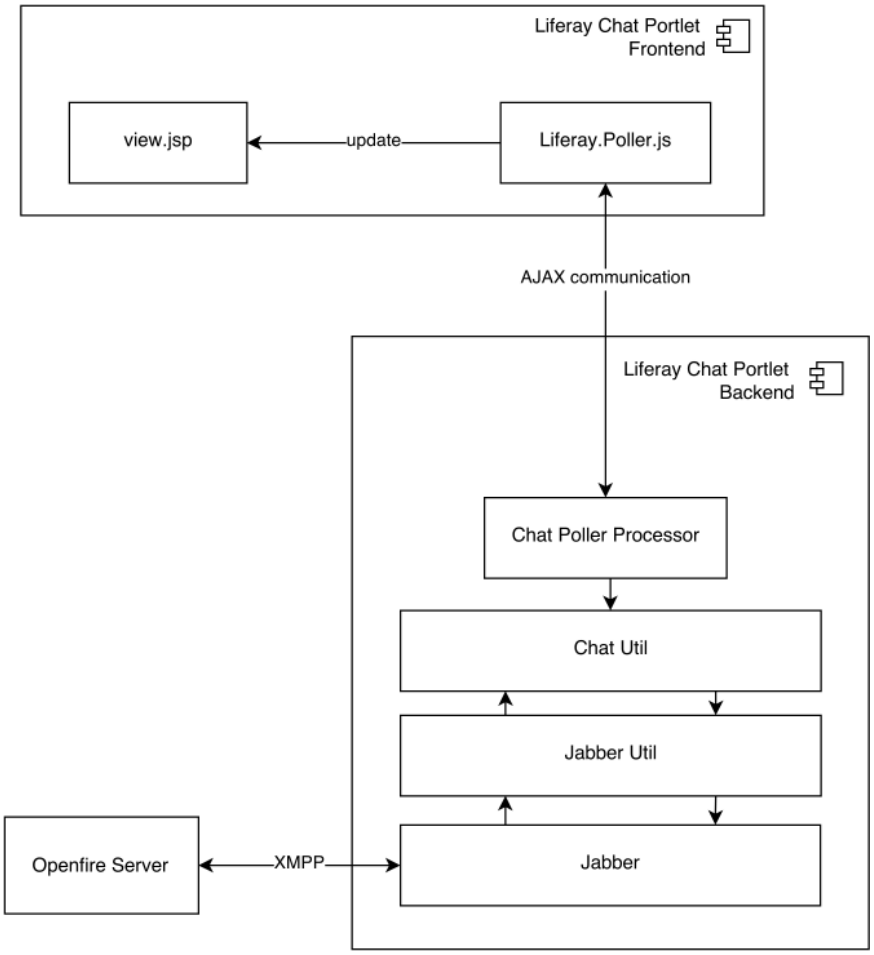

Figure 1.3: Liferay Chat Portlet infrastructure

1.10.1 Communication

Liferay Chat Portlet sends two types of AJAX requests:

- *receive* – frontend sends an AJAX request every 4-8 seconds to check if there is any change. Chat Poller Processor responds with a JSON object containing a list of updated data (Figure 1.4). There is no

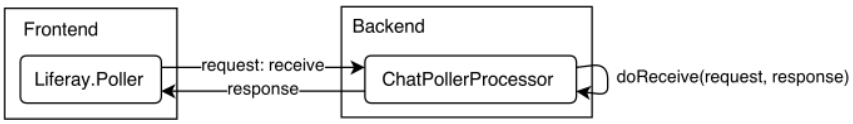

Figure 1.4: Poller Receive Request

way to explicitly trigger the receive action. Liferay has its own timer, which repeats the given operation for each allotted amount of time.

The programmer usually starts poller. Liferay.Poller object's method *addListener*, also called by the programmer, is served with a parameter specifying a pointer to the custom callback method, which is called whenever the poller receives a response.

- *send* – on the other hand, a way to send data to the backend side still exists (Figure 1.5).

  Liferay.Poller has a method called *submitRequest(portletId, data, key)*. Unfortunately, this method does not return any value. As a result,

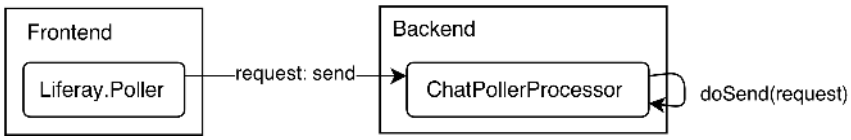

Figure 1.5: Poller Send Request

there is no way to confirm success or failure of the given request because no response (!) is received. Consequently, there is no other way to receive data from the backend besides waiting for the response of the receive action.

1.10.2 Frontend

The Frontend mainly consists of javascript components and a view page. Although the javascript code is located in a single *main.js* file, it is divided into separate classes. For future development, it would be necessary to split the classes into separate files for the purpose of clarity.

1.10.3 Backend

The Backend has the following responsibilities:

- Request handling,
- User session management,
- List distribution of connected users,
- Message delivery,
- Jabber server integration.

User session management

Backend uses Liferay Hooks [37] to perform custom actions based on circumstances that might occur:

- *LoginPostAction* [40] – to obtain user credentials and login to the Jabber server.
- *SessionDestroyAction* – to perform disconnect action on the Jabber server.
- *UserListener* – to remove any chat data related to the user who is being removed from the Liferay database.

Jabber server integration

One of the biggest advantages of the Liferay Chat Portlet is the support for Jabber server integration. As described in documentation: "Jabber is the original name of the XMPP (Extensible Messaging and Presence Protocol) protocol, an open-standard communications protocol based on XML. Using a chat server helps Liferay's chat scale to very large installations and allows for communication between different chat clients."[44]

Jabber server integration is not enabled by default since it requires a running Jabber server. It can be enabled in the *portlet-ext.properties* by setting the *jabber.enabled* value to true. If the integration is enabled, the backend distributes all messages to the Jabber server but also stores them to the Liferay database. This behaviour may cause security issues. Also, it is not necessary to store the same information on two separate databases.

Communication with the Jabber server is mutual. User A, who is using Liferay Chat Portlet, may send a message to the user B, who is using another Jabber client connected to the Jabber server and vice versa. However, only the one-to-one communication is supported. As a result, no conversation can have more than two participants.

Future improvement

Liferay Chat Portlet does not fulfil some of the project goals:

- Support for multi-user chat,
- Precise description of status (e.g. on-line, busy, unavailable),
- Possibility to turn the chat off.

Also, several issues were discovered during research:

- Open conversations disappear when the user moves to different page,
- User interface does not match University Information System design,
- Pure code quality,
- No documentation,
- Business logic in scriptlets.

More requests for improvement are listed here [34] and are also a part of Liferay's JIRA [45].

Overview

Current stable version: 1.0.2
Requirements: Liferay Portal 6.1 CE GA2+
Programming language: Java
Licence type: LGPL

Chapter 2

Analysis and Design

2.1 Functional requirements

1. *Authentication via LDAP directory*
   User credentials will be authenticated via LDAP directory during the login process to Liferay portal.

2. *Concept of private conversations*
   Users will be able to create their own private conversations. The creator of a conversation will have the option to choose participants during the creation process. Afterwards, the creator becomes a regular participant. There is no need to explicitly distinguish between the conversation owner and a regular member (although the XMPP protocol has a direct support for this). Any participant will be able to add other users to the conversation. Users not added by other participants cannot become a participant. Any participant can leave the conversation at any time.

3. *Concept of public conversations*
   Users will become participants in public conversations based on several circumstances. For example, if the student is signed up in the Math class 2013 he will become a participant in the public room Math 2013. The given approach will lead to tighter social connections. Accordingly, students will have the option to discuss the class and share their opinions with their schoolmates or even teachers in the automatically created groups. Currently, two public rooms should be created:

   a) *Alumni Bachelor* - public room containing users who graduated with a bachelor degree

   b) *Alumni Master* - public room containing users who graduated with a master degree

4. *People in conversation*
   Each conversation will have a list of participants and their current status. As a result, each participant can quickly check if the other participants are able to see his/her messages.

5. *User status*
   Users will have the option to decide their current status (e.g. online, busy, unavailable or offline).

6. *Possibility to turn the chat off*
   If the user does not want to communicate with the others, the option exists to turn chat off. The reverse should also be true to turn the chat function on again.

7. *New message notification*
   Users will be notified about incoming messages via:

   a) extension of page title with the number of unread messages,

   b) badge above particular panel with a number of unread messages,

   c) audio notification.

   The user will have the option to switch the audio notification off.

8. *Adding other participants to the conversation*
   the user will have the option to add other participants to the conversation.

9. *Leave conversation*
   The user will have the option to leave the conversation.

10. *GEXF file generation*
    The system will be able to generate a GEXF file with the information about social ties between participants.

2.2 Project architecture

The project consists of several independent components with strictly defined responsibilities. All of them have to communicate with each other extensively in order to create the illusion of a real-time communication (Figure

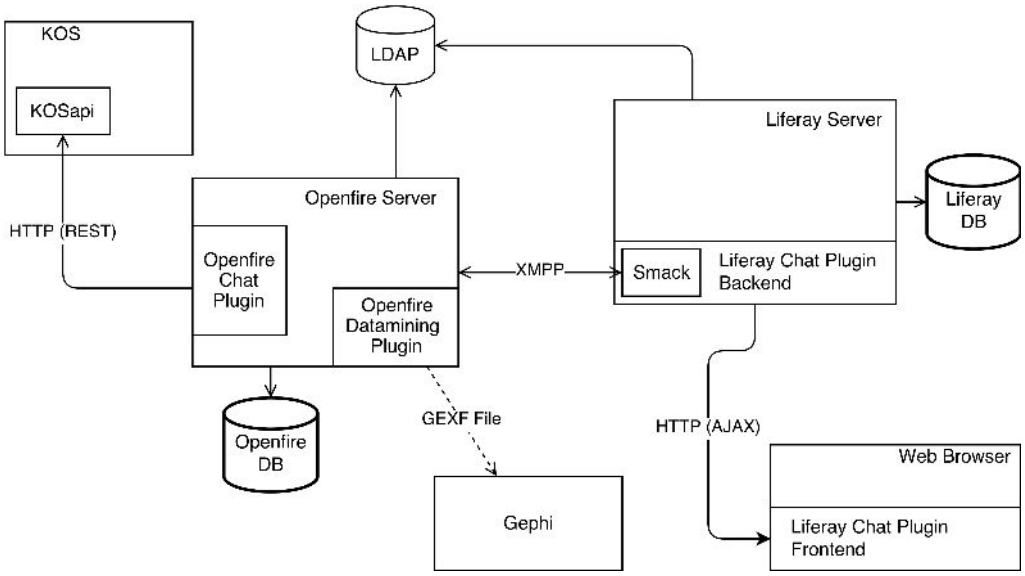

Figure 2.1: Architecture

2.1). The following sections describe an overview of the project architecture, its components, their responsibilities and the way they communicate.

Communication between the components is based on the following protocols: XMPP, HTTP (AJAX and REST). Communication between Openfire Data Mining Plugin and Gephi is based on the file transfer. The plugin generates a single XML file, which can be opened via Gephi.

2.2.1 XMPP protocol

XMPP protocol is the main communication protocol between the Liferay server (Smack API particularly) and the Openfire server. The definition of the entire protocol is beyond the scope of this thesis. On the other hand, a basic understanding of some principles is necessary for further reading.

JID

Jabber ID (usually abbreviated as JID) serves as a unique XMPP identifier. It is made up of a node (generally a username), a domain, and a resource. The node and resource are optional; the domain is required. In simple ABNF[1] form:

```
jid = [ node "@" ] domain [ "/" resource ]
```

[1] ABNF – Augmented Backus–Naur Form

Some sample JID's:

```
user@example.com
user@example.com/home
example.com
```

Each allowable portion of a JID (node, domain, and resource) must not be more than 1023 bytes in length, resulting in a maximum total size (including the '@' and '/' separators) of 3071 bytes.

Resource

XMPP protocol supports multiple clients via *resources* [55]. As a result, there is no need to have different accounts while logged in via different clients. Each client may have its own resource identifier. For example, if the user John Doe with account *john.doe@example.com* signs up via Liferay, his JID will be:

`john.doe@example.com/Liferay`

Similarly, if he signs up via the other client, e.g. Pidgin the resource may look like:

`john.doe@example.com/Pidgin`

Multi User Chat

Initially, instant messaging was meant to be a one-to-one rather than a many-to-many chat. However, in 1999 Jabber group developed a multi user chat protocol [52]. Since then, it has been possible to have a conversation with multiple participants.

Service

XMPP extension XEP-0045: Multi-User Chat describes service as: "A host that offers text-based conferencing capabilities; often but not necessarily a sub-domain of an XMPP server (e.g., conference.jabber.org)."[52] It serves as the hostname on which the multi-user chat service is running.

Room

A room is a virtual space that users figuratively enter in order to participate in real-time, text-based conferencing with other users.

Room has a unique identificator called “room JID” which is in a form of:

```
room@service
```

where *room* is the name of the room and *service* is the hostname at which the multi-user chat service is running.

Some sample Room JID’s:

```
teachers@conference.example.com
alumni@conference.example.com
```

Each occupant in a room is identified with an “occupant JID” in a form of:

```
room@service/nick
```

where *nick* is the room nickname of the occupant as specified on entering the room or subsequently changed during the occupant’s visit.

Some sample Occupant JID’s:

```
teachers@conference.example.com/john.doe
alumni@conference.example.com/peter.griffin
```

2.3 Openfire Server

Openfire is the heart of the whole architecture. It is responsible for the following:

- List maintenance of private and public conversations,
- Storage of history,
- Management of user statuses,
- Message delivery,
- User session maintenance.

To fulfil project goals, its functionality will be extended via the Chat Plugin and Data Mining Plugin. The first one will assure correct synchronization between virtual groups on the KOS server and public rooms on the Openfire side. The second one will create a Gephi compatible XML file based on the user chat history.

2.3.1 Openfire plugin development

Plugins enhance the functionality of Openfire. Additionally, the developer is able to access the internal API of the server which includes access to the user or room management, history, handling of privileges and many other useful features.

Each plugin has to maintain a predefined structure [29]. Moreover, plugin icons and metadata files like *readme.html*, *changelog.html* and *plugin.xml* must be included too.

The plugin.xml file is necessary for the description of plugin. It contains definition of the main plugin class that will be called whenever the plugin is initialized. Other metadata tags e.g. name, description, author, version, date, url, minimal server version and license type can be set here too. They usually serve as descriptive information in the plugin section of the admin user interface. Thus it is good practice to complete them before taking the plugin public.

Another important plugin.xml function is to define behavior of the admin console, especially the left side bar. Each time the plugin is deployed it appears as a link in the Server Settings menu. From the plugin.xml file it is possible to define its id, name, description and url.

A plugin can be built and deployed via the Openfire build script. The build script will compile source files and create a valid plugin structure and JAR file, which is the only file needed during the deployment process.

2.3.2 Openfire Chat Plugin

The main responsibility of this plugin is to assure the correct synchronization between virtual groups on the KOS server and public rooms on the Openfire side. Currently, there is a requirement to assure the synchronization of:

- *Alumni Bachelors* – users who graduated with the bachelor degree and are no longer students of the University.
- *Alumni Masters* – users who graduated with the master degree and are no longer students of the University.

Synchronizer

The synchronization process is handled by the Public Room Synchronizer (Figure 2.2). The given object inquires Kos Wrapper about the list of

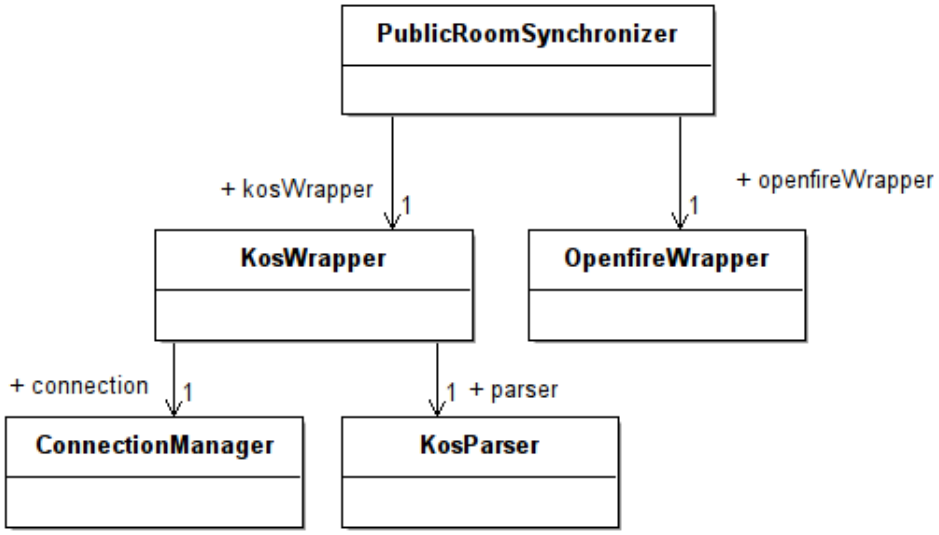

Figure 2.2: Openfire Chat Plugin class diagram

Alumni Bachelors and Alumni Masters. Afterward, the given list is distributed to the Openfire Wrapper which creates related public rooms.

KOSApi

The Openfire Chat Plugin retrieves data from KOS. KOSapi [22] serves as a bridge between KOS and the rest of the world. It provides an API in a form of RESTful web service. The REST call, which returns a list of alumni bachelor users is in this form:

```
/students?query=
studyTerminationReason==GRADUATION;
faculty=18000;
programme.type=BACHELOR
&limit=500
```

Similarly, for the list of alumni master users:

```
/students?query=
studyTerminationReason==GRADUATION;
faculty=18000;
programme.type=MASTER
&limit=500
```

Kos Wrapper encapsulates the logic which is needed to obtain data. The Connection Manager serves as a connector to the KOSapi service. Kos Parser analyzes the retrieved data in XML format and creates the appropriate objects.

2.4 Openfire Data Mining Plugin

The main function of the data mining plugin is to generate a GEXF file based on the data from private conversations. GEXF [20] is a language for describing complex network structures, their associated data and dynamics.

The Openfire Data Mining Plugin monitors communication between users. Furthermore, based on the obtained data, it constructs complex networks, which are described via the GEXF file. It might then be opened in any editor which is compatible with the given file format (e.g. Gephi).

2.4.1 Network construction

To illustrate network construction, imagine the following scenario. Angela, Brian and Chloe are together in a conversation (Figure 2.3). Angela sends a message to Brian and Chloe. Message is wrapped in the envelope object that contains information about the sender (from), recipient (to) and a total number of messages that were exchanged between these two participants in all conversations (count). The graph which is generated from the previous communication has three nodes (A, B, C) and two edges (A-B, A-C).

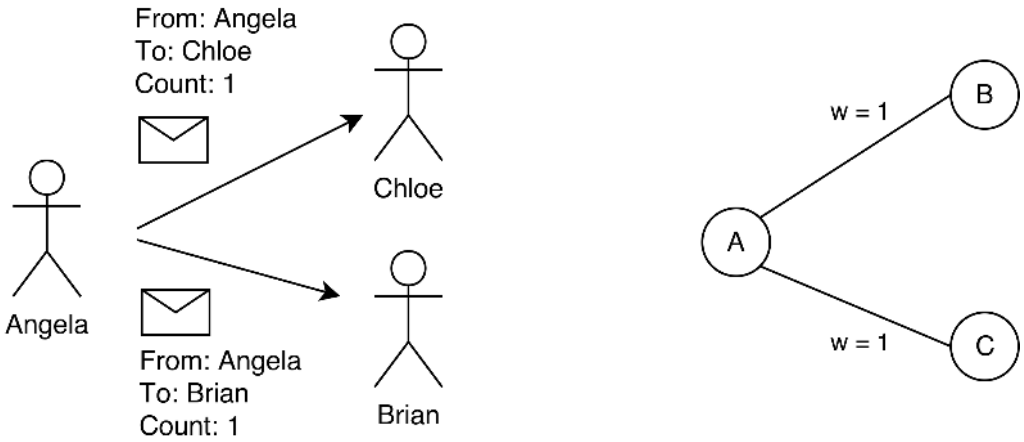

Figure 2.3: Angela sends a message to Brian and Chloe

Each edge has a weight parameter, which represents the count of messages exchanged between two nodes. Higher weight is visualized by a thicker line.

Later on, Angela sends another message (Figure 2.4). As we can see in the picture below, the envelopes remain. Only the number of messages has been incremented. Subsequently, the weight parameter is higher and the lines between nodes are thicker.

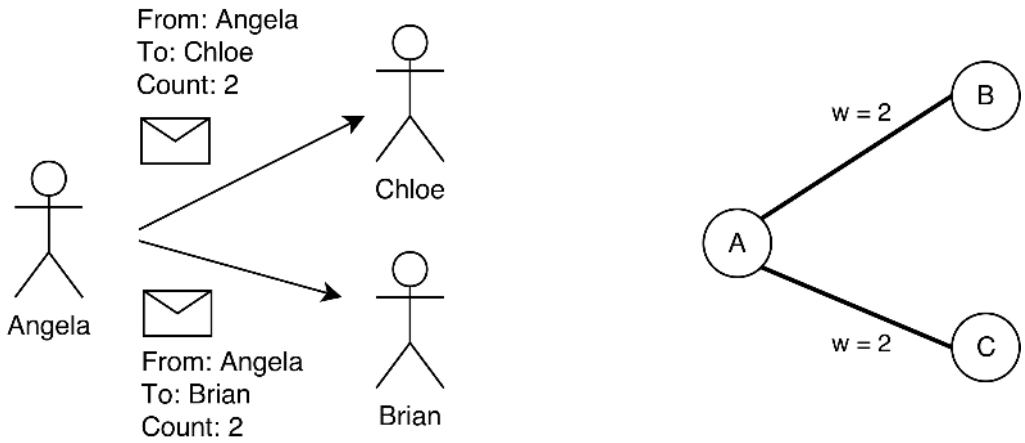

Figure 2.4: Angela sends another message to Brian and Chloe

In the meantime, Angela and Chloe can have their own conversation without Brian. Angela sends a message to Chloe (Figure 2.5). As expected, the message count is incremented as well as the weight parameter. Moreover, the line between nodes A and C is thicker but the line between A and B remains.

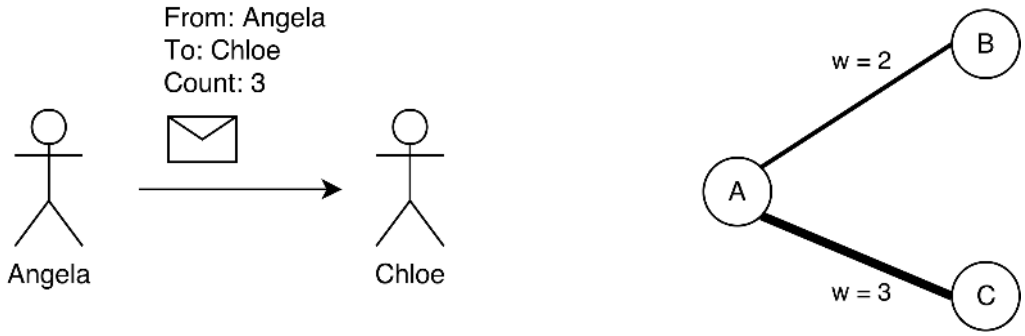

Figure 2.5: Angela sends a message to Chloe

Last but not least, Chloe sends a message to Angela (Figure 2.6). In such a case, a completely new independent envelope is created. The envelope

creation is thus bidirectional. On the other hand, the graph does not distinguish between these two envelopes and simply sums the count[2].

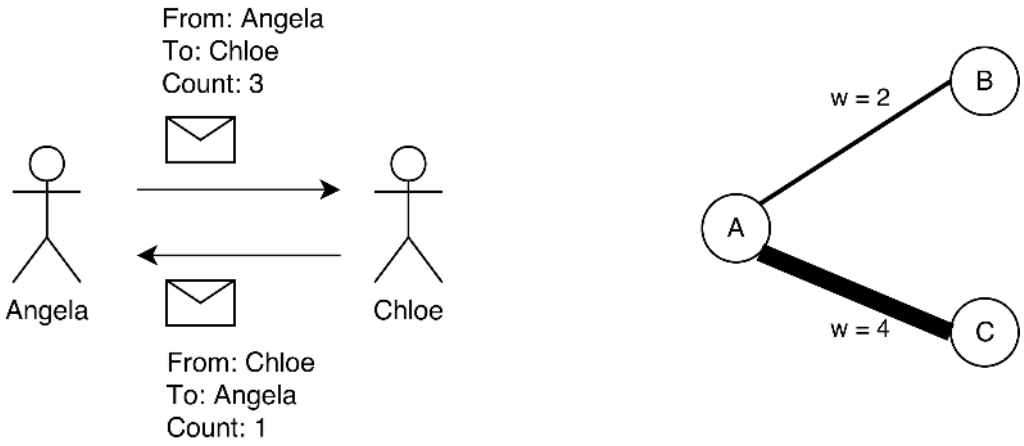

Figure 2.6: Chloe sends a message to Angela

2.4.2 Generator

The generator object manages generation process (Figure 2.7). Firstly, it collects all private rooms within the system from the Openfire Wrapper. Furthermore, it constructs envelopes based on the chat history from the private rooms. Finally, it passes the list of envelopes to the Gephi Wrapper object which creates a GEXF file.

2.5 Liferay Chat Plugin

Liferay Chat Plugin serves as a client to the Openfire server (Figure 2.8). It is divided into the backend and frontend side. The backend side is responsible for communication with the Openfire server and also for view creation. The frontend side handles all user actions and communicates with the backend via the AJAX calls. Liferay disposes of the own poller mechanism, which was already described in the Liferay Chat Portlet section.

[2]This is not a dogmatic rule. Data Mining Plugin can be always modified to accept bidirectional generation of network. It was simply not considered to be useful during the analysis phase.

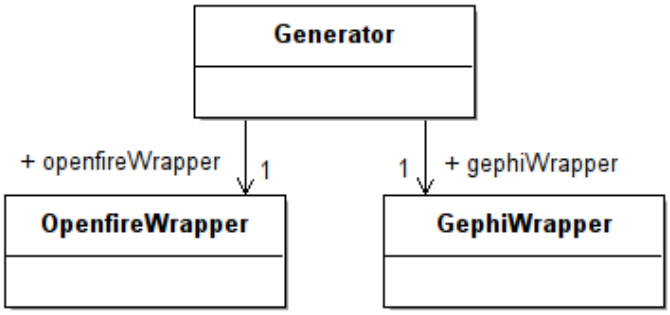

Figure 2.7: The generator object manages generation process

2.5.1 Backend

The backend is divided into four layers (Figure 2.8):

- *Data Transfer Layer* – contains *Chat Portlet* and *Chat Poller Processor* objects. It is responsible for view creation during the initial request and for communication with the frontend.
- *Chat Util Layer* – serves as an API to the Liferay Chat Plugin functionality. All requests must be triggered via this layer. No objects within the plugin can be accessed directly. It also serves as a bridge between the transfer layer and jabber util layer. As a result, the transfer layer is loosely coupled with the actual jabber implementation. Due to this fact, it is possible to change the jabber util layer with no effect on the transfer layer side.
- *Jabber Util Layer* – serves as an API to the actual jabber implementation.
- *Jabber Layer* – contains the actual jabber implementation. It communicates directly with the jabber server.

Liferay Chat Portlet has all the server logic located in scriptlets within the *view.jsp* file. This is not considered as a good programming practice. Moreover, there does not exist any controller class either. The server logic should be a part of the separate controller class and view should be responsible for the representation of data. Therefore, Liferay Chat Plugin implements separate Chat Portlet class which servers as a controller and is responsible for the creation of View, which is in a form of JSP[3] page.

[3]JSP – Java Server Page

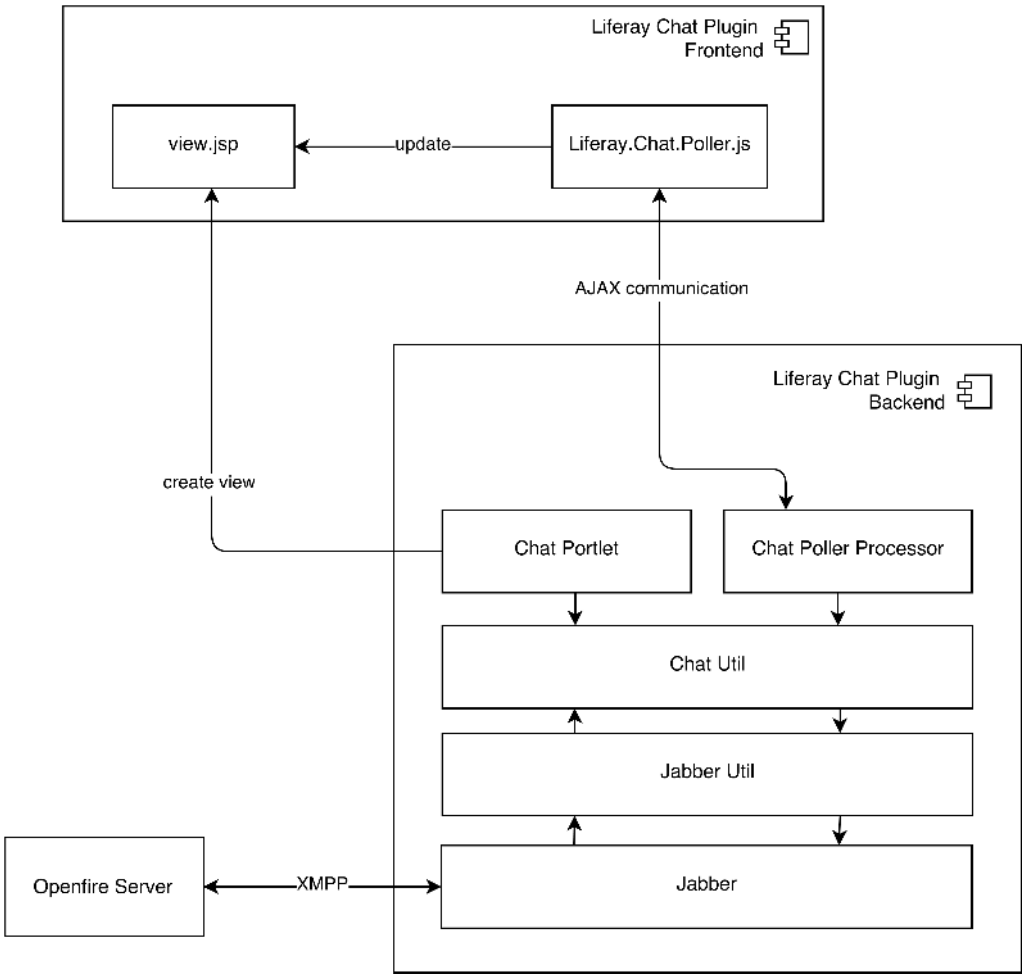

Figure 2.8: Liferay Chat Plugin architecture

Conversation Container

Whenever the user logs in, the backend creates a Conversation Container that contains all conversations in which the user participates. Furthermore, it instantiates proper Conversation objects. The user is set as the owner of each object (owner relationship on the Figure 2.9). Each conversation stores all messages and participants that belong to it. Finally, it has a link to the proper jabber room.

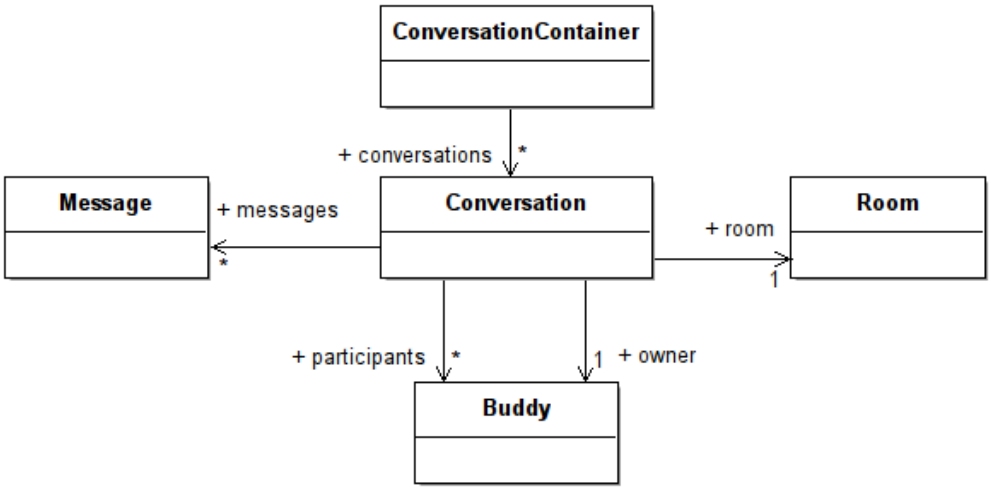

Figure 2.9: Liferay Chat Plugin conversation model

Database model

Some of the information needs to be stored in database (Figure 2.10). Each user participates in several rooms and some of them are opened – visible on the panel. Also, each user has a list of settings that belong to their account.

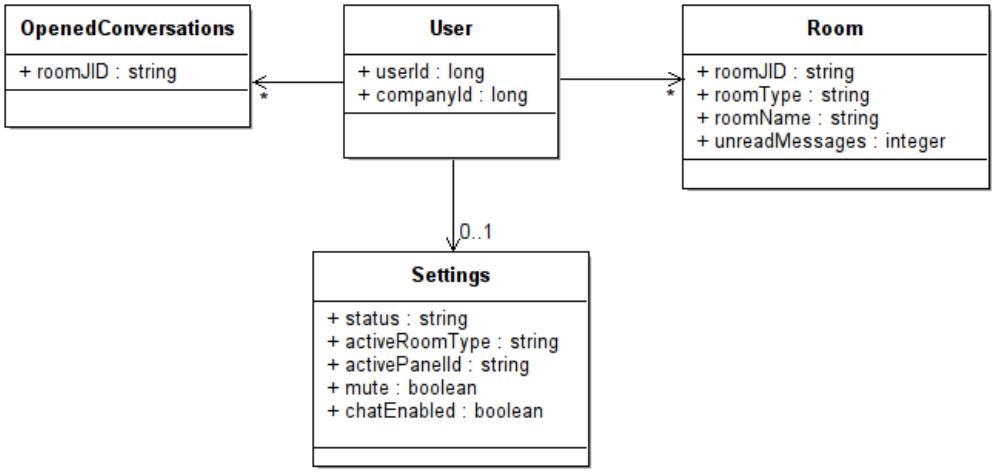

Figure 2.10: Database model

Room

The room table contains the following columns:

- *roomJID* – unique jabber id for the given room;
- *roomType* – describes room type. it has two possible values:
  - jabber.room.type.private
  - jabber.room.type.public
- *roomName* – public rooms usually have a name which is stored here;
- *unreadMessages* – number of unread messages within the conversation that belong to the given room.

Opened Conversation

The opened conversation table contains this column:

- *roomJID* – unique jabber id for the room that belongs to the open conversation.

Settings

The settings table contains the following columns:

- *status* – indicates current user's status. Can have the following values:
  - jabber.status.online
  - jabber.status.busy
  - jabber.status.unavailable
  - jabber.status.off
- *activeRoomType* – indicates the tab which is selected by user in the conversation list panel. There are two possible values:
  - jabber.status.public
  - jabber.status.private
- *activePanelId* – indicates active (open) panel;
- *mute* – set to 1 if the user does not want to be notified about the incoming messages by playing a sound, 0 if otherwise;
- *chatEnabled* - set to 1 if chat plugin is enabled, 0 if otherwise.

2.5.2 Frontend

The frontend consists of multiple components, the manager object and a view page. Firstly, the view page is rendered. At the end of the rendering process, while all components are loaded, the manager object takes care of the orchestration.

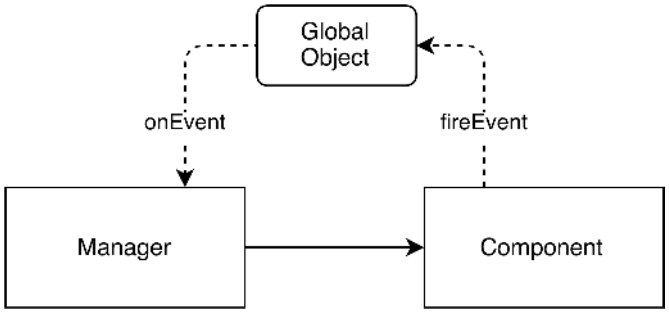

Figure 2.11: Fire an event via the Global Object

The components never call the manager directly (Figure 2.11). They always fire an event to the Global Object [57]. The manager then listens to all events which might occur. As a result, the components are loosely coupled with the manager. They only fire an event and do not care about consequences.

The manager points to several objects (Figure 2.12). First of all, Poller is responsible for the server side communication (backend). No other objects can use poller directly. They always fire an event that is captured by the manager, which then uses poller if needed. The notification object is responsible for the sound notifications on incoming messages, and the error object shows or hides error messages. The manager also points to several containers, which are responsible for the panel rendering.

Panels

The frontend view layer mainly consists of panels. A panel is an independent visible area, which can be created statically from an already rendered html markup or dynamically based on the user action or poller request. It has a title, content and buttons that trigger display actions like hide, show, toggle and close (Figure 2.13).

The panel is automatically set to the *hidden* state whenever it is created. While the panel is *hidden*, only the panel trigger is visible. If a user clicks on the trigger, the panel changes state to *displayed* (toggle action). Concurrently, other panels are set to hidden. As a result, only one panel

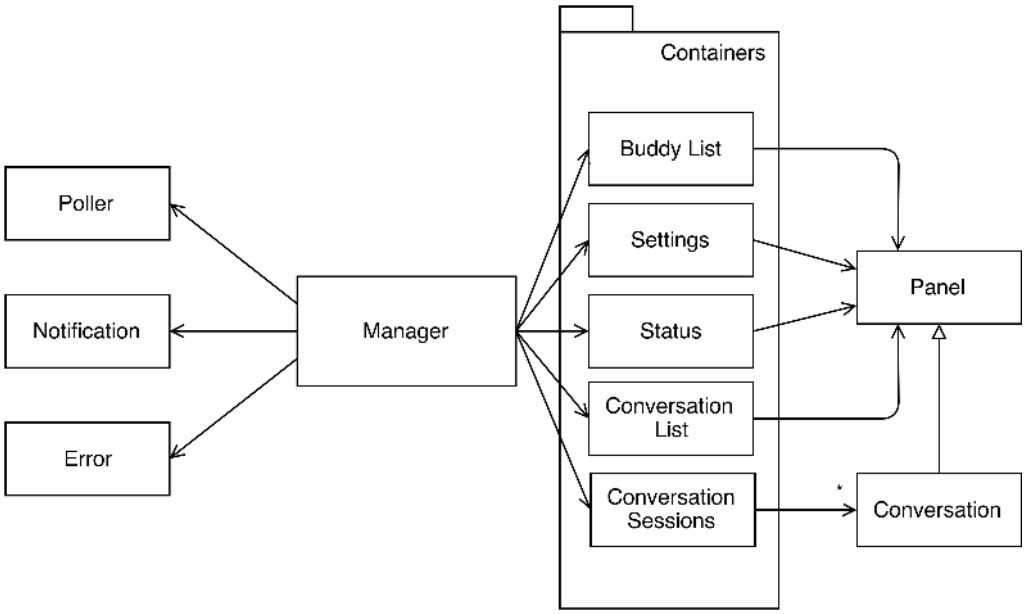

Figure 2.12: Frontend architecture

can be displayed. Each visible panel consists of panel title, content, buttons and trigger. The trigger always rotates between the *hidden* and *displayed* state. The hide button hides the displayed panel and the close button sets the panel to the *closed* state. The closed panel is destroyed and removed from the container.

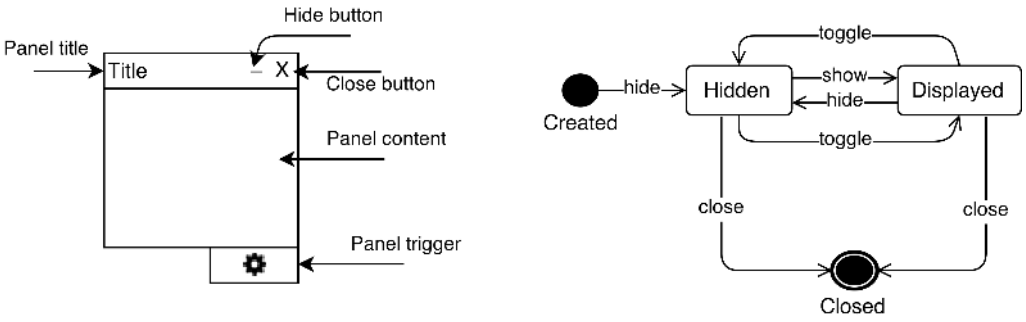

Figure 2.13: Frontend panel

Containers

The containers (Figure 2.14) encapsulate logic that is needed to handle related panels. They also listen to the user actions and fire proper events to the Global Object.

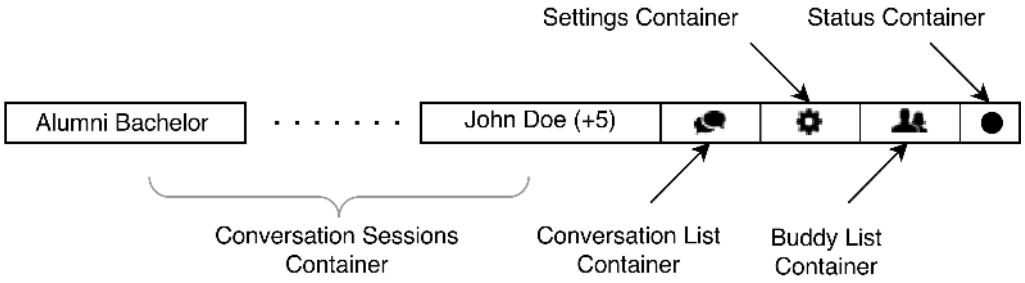

Figure 2.14: Frontend containers

Status container

The status container controls the status panel (Figure 2.15) where user may select his/her status or turn chat off. The status container fires the following events:

- *statusUpdated* – whenever user changes status.

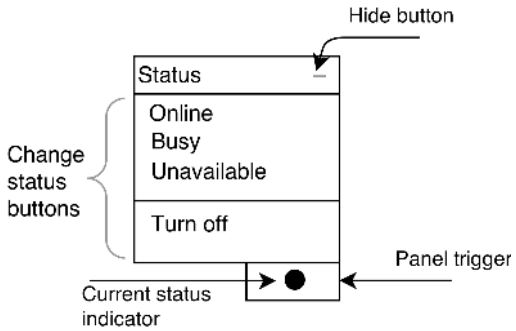

Figure 2.15: Status panel

Buddy list container

The buddy list container controls the buddy list panel (Figure 2.16). The given panel contains a list of all users who participate in conversations

where logged users participate too. Each line displays the user's portrait, full name and his/her current status. The List can be filtered via the search bar.
The container fires the following events:

- *buddyListUpdated* - whenever the list of buddies is updated.

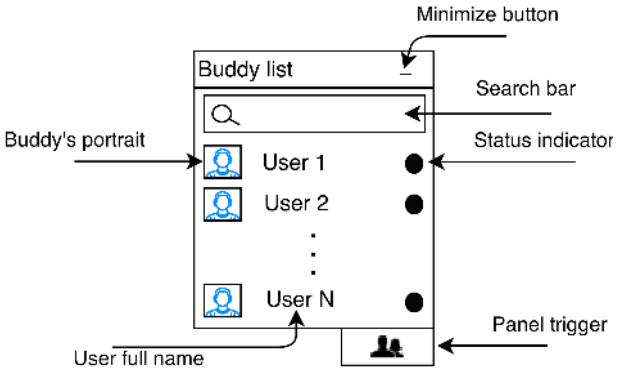

Figure 2.16: Buddy list panel

Settings container

The settings container controls the settings panel (Figure 2.17). The given panel contains a list of all possible settings within the application. Currently there is only one option; to switch the notification sound on/off. The container fires the following events:

- *settingsUpdated* - whenever the user changes any setting.

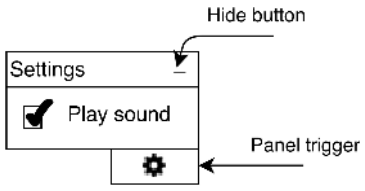

Figure 2.17: Settings panel

Conversations container

The conversation container has two responsibilities. It controls the panel that shows a list of conversations where the user participates. Furthermore, it provides a functionality to create a new conversation (Figure 2.18).

The panel contains both private and public conversations. It is possible to switch between them thanks to the *conversation type switch*. Each row of the conversation list contains a portrait of the last message sender, a list of participants and the last message. The new conversation button opens a new conversation menu.

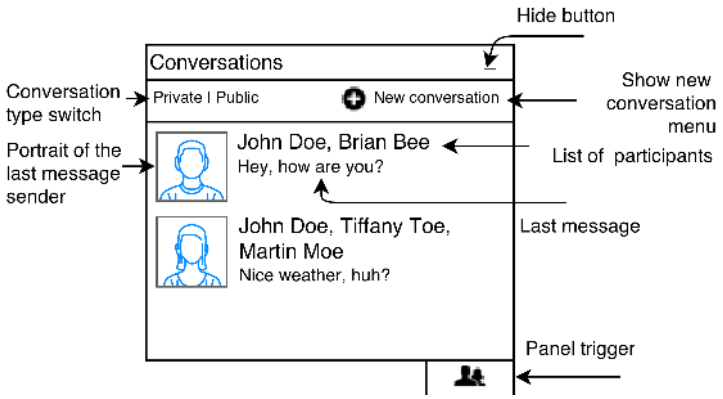

Figure 2.18: Conversation list panel

The new conversation menu contains (Figure 2.19) a list of participants. Initially, the list is empty. Whenever the user starts typing, the system shows a dropdown list of possible buddies[4], from which the user may select. A participant may be removed from the list by clicking on the remove from list button next to his/her name. A new conversation cannot be created if there is no buddy on the list or if the message box is empty, and the system warns the user whenever he/she wants to create a new conversation with either an empty participant list or an empty message box. The container fires the following events:

- *newConversationCreated* – whenever the user clicks on the send button (participant list and message box are not empty).
- *conversationSelected* – whenever the user clicks on a conversation in the conversation list.

[4]Othe users within the system

- *publicConversationSelected* – whenever the user clicks on the public conversation switch.
- *privateConversationSlected* – whenever the user clicks on the private conversation switch.

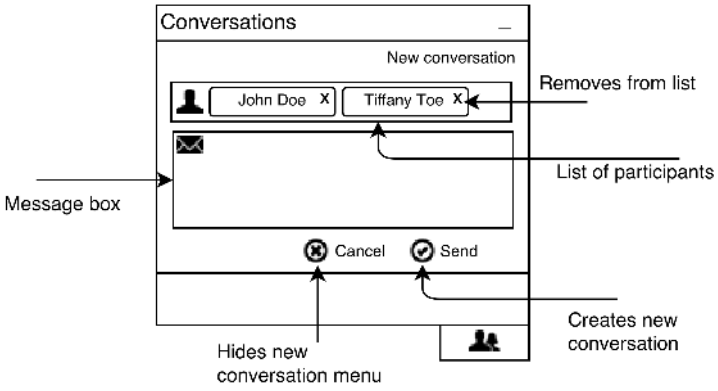

Figure 2.19: New Conversation menu

Conversation sessions container

The conversation sessions container takes care of opened conversations. The conversation is an extended panel (Figure 2.20) and its main responsibility is to show the message feed. Messages are sorted chronologically. The oldest are at the top and the newest at the bottom. Each message contains the participant's portrait, his/her full name and a message timestamp which shows the difference between the current time and the time when the message was sent (e.g. 5 mins ago).

The top part of the conversation panel contains the portrait and full name of the last message sender. Moreover, it contains several buttons such as open menu, hide and close. The search bar is a part of the search menu, which will be described later. If the user clicks on the close button, the conversation disappears. To make it visible again, the user must click on the given conversation in the conversation list panel.

The conversation panel also holds the following menus:

- *Search menu* – allows user to search for any phrase within the message feed;
- *Add to conversation menu* – adds new participants to the conversation;

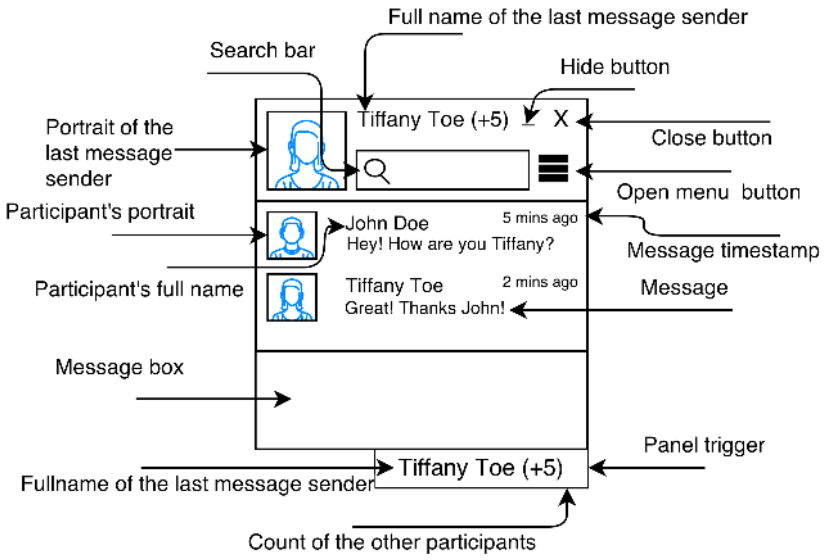

Figure 2.20: Conversation panel

- *People in conversation menu* – displays participants within the conversation;
- *Leave conversation menu* – allows user to leave the conversation.

2.5.3 Liferay Chat Portlet Extensions

The previous research revealed several missing features in Liferay Chat Portlet:

- Support for multi-user chat,
- Precise description of status (e.g. on-line, busy, unavailable),
- Possibility to turn chat off,
- Opened conversations disappear when user moves to different page.

Based on the given circumstances, there is a need to extend the existing functionality with the foregoing features. This will be achieved with the Liferay Chat Plugin.

Multi User Chat Support

All conversations will be considered many-to-many conversations, even with only two participants in the conversation. This will make it easier to add more participants to the conversation.

Precise description of status

The user's presence is described by the following status types:

- *online* – user is actively interested in chatting;
- *busy* – user is busy but still interested in chatting;
- *unavailable* – user is not interested in chatting but available in the urgent situations;
- *off* – user is not logged in or he/she turned chat off.

XMPP protocol disposes of even more statuses. However, the aforementioned set of statuses was considered sufficient. The table 2.1 shows the mapping between the XMPP and Liferay Chat Plugin statues.

Table 2.1: Status conversion table

Liferay Chat Plugin Status	**XMPP Status**
online	available
busy	away
unavailable	dnd
off	unavailable

Possibility to turn the chat off

The Liferay Chat Portlet cannot be turned off. The absence of this feature, as the chat portlet is visible on each portal page, might be considered unpleasant for the user's experience. Therefore, the Liferay Chat Plugin will have an option to turn the chat off. Accordingly, if the chat is off no other panels apart from the status panel will be visible.

However, the connection between the jabber server and chat plugin cannot be disconnected because the user password is only reachable during the login process. Hence If we disconnect from the jabber server there is no

possibility to reconnect again. Due to this fact, turning the chat off will not trigger a disconnect action. It only sets user XMPP status to unavailable. Consequently, the user will still be connected to the jabber server but he/she will not be shown as available.

Opened conversations disappear when user moves to different page

Whenever the user goes to another page within the portal, all opened conversations disappear for a while. This is caused by a small gap[5] between the create view action and the initial response from the Chat Poller Processor. To avoid this behavior, all panels on the frontend need to be rendered during the create view action. On the other hand, the message feed may be relatively long (thousands of messages). Moreover, the user may have multiple conversations open. If the server loads all messages during the create view action it may slow down server responsiveness. Given these facts, the message feed will be delivered at the initial request by the Chat Poller Processor after the create view action (Figure 2.21).

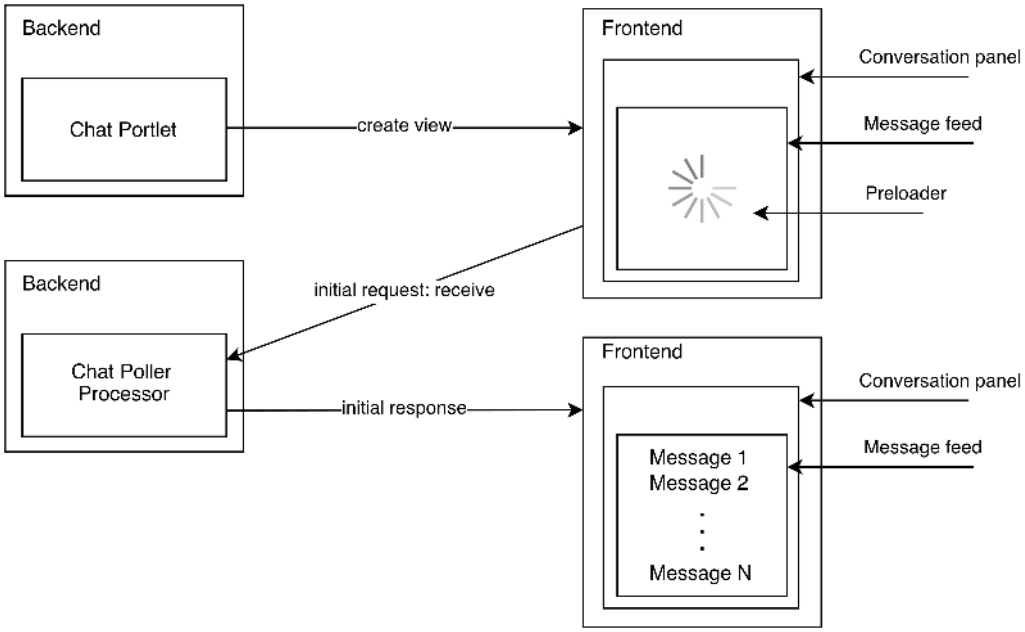

Figure 2.21: Message feed loading

[5]Usually less than 5 seconds.

Due to this fact, the message feed does not contain messages until the frontend receives initial response. This behavior may cause confusion, which will be avoided by displaying the preloader until the message feed loads.

Chapter 3

Implementation

The following lines introduce the project from the implementation perspective. The analysis and design from the previous chapter described a general view of the problem. From now on the focus will be on a specific implementation of each component.

3.1 Openfire Chat Plugin

The Openfire Chat Plugin synchronizes users from the virtual groups on the KOS server with users in Openfire public rooms. To provide the given functionality, it needs to collect the following information:

- *Service name* – name of the service all public rooms belong to;
- *Public room prefix* – to distinguish between public and private rooms;
- *KOSapi URL* – resource locator for the KOS REST service;
- *KOSapi Username* – login username for the KOS REST service;
- *KOSapi Password* – login password for the KOS REST service;
- Name of the alumni bachelor public room;
- Name of the alumni master public room.

The values are collected via the web form located on the Chat plugin settings page. It also contains a button that triggers synchronization with KOS virtual groups.

The plugin consists of two servlets:

- *Settings Servlet* – responsible for the settings form;
- *Kos Synchronization Servlet* – responsible for the synchronization process.

Servlets are defined in the *web-custom.xml* file. The given file also contains servlet mappings to the url patterns. For example, the Settings Servlet is mapped to the */settings* url pattern and thus accessible at:

```
http://url.domain/settings
```

Settings Servlet

The Settings Servlet is responsible for the collection of properties from the users via the web form. All properties are stored in the Openfire database and further accessible via the Plugin Properties object. All properties are thus persistent and should remain after the crash or restart of the server.

Kos Synchronization Servlet

Asynchronous requests from the Settings Servlet are handled via the Kos Synchronization Servlet. Based on good coding standards, all the business logic is located outside of the servlet itself in a separate class, the Public Room Synchronizer, which takes control of the synchronization process (Figure 3.1). At the end, the servlet sends a response with a JSON Object that contains information about the success or failure of the process (Figure 3.2). The Javascript code on the frontend side is thus able to show the possible success or error message.

Public Room Synchronizer

The Synchronizer takes responsibility for an orchestration of two objects - Kos Wrapper and Openfire Wrapper. Basically, it requests data from the first and then passes it to the second. If there is any problem during the process it notifies Kos Synchronization Servlet about the failure by throwing an exception (Figure 3.2).

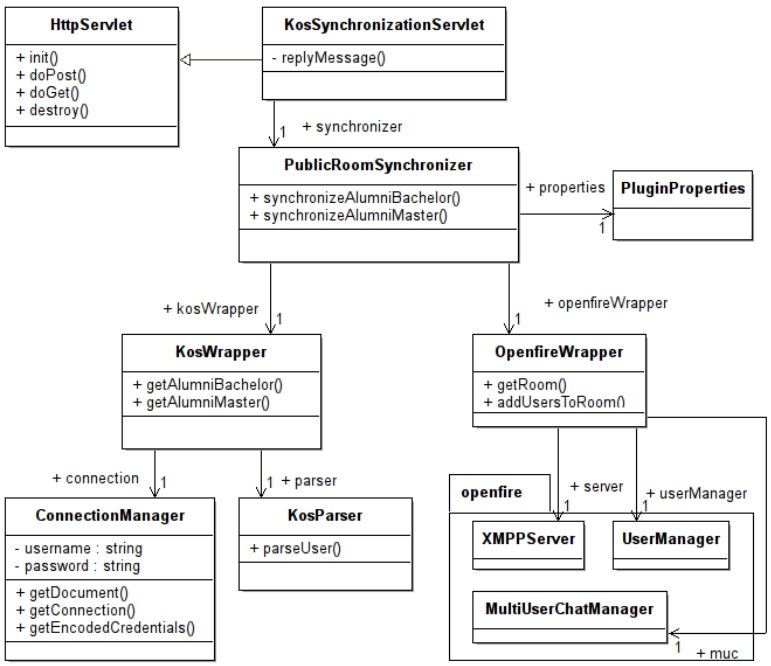

Figure 3.1: Kos Synchronization Servlet class diagram

Kos Wrapper

All logic needed to handle the connection with the KOSapi service is located within this object. It requests data from the KOSapi service by the REST call and then parses the response and creates a list of users.

Figure 3.2: Kos Synchronization Servlet sequence diagram

Openfire Wrapper

Any communication with the Openfire server is maintained via this class. During the process of synchronization it takes a user list, room JID and room name. If there is no room with the given JID, it creates a new one and sets its name based on the parameter it receives.

3.2 Openfire Data Mining Plugin

Openfire Data Mining Plugin generates GEXF, which contains the data needed for the social network analysis. Data is generated based on the private conversations. The plugin thus operates with the Gephi and Openfire Wrapper classes. The generation process is controlled by the Generate Servlet class.

Generate Servlet

The generator servlet has two responsibilities. Firstly, if a GET request is received, it forwards to it the generate.jsp page. Secondly, if it receives a POST request, it generates a GEXF file via the Generator object. The Generator does not actually generate a file which gets saved in the file system and distributed to user. Conversely, it writes the data directly into the response output stream. To make it possible, the following lines of code need to be added to the response to set a different content type and change a response header:

```
response.setContentType("application/octet-stream");
response.setHeader("Content-Disposition",
"attachment; filename=graph.gexf");
```

As a result, the web browser automatically takes the output stream and saves it as a graph.gexf file in the user's file system.

Generator

Whenever the Generator Servlet (Figure 3.3) calls the *generate()* method on the Generator object, it receives an output stream which contains the desired graph. To make it possible, the Generator takes a list of private rooms from the Openfire Wrapper object. Afterwards, it iterates through the list and creates envelopes. The envelope list is then passed to the Gephi Wrapper object, which constructs and returns the final graph (Figure 3.4).

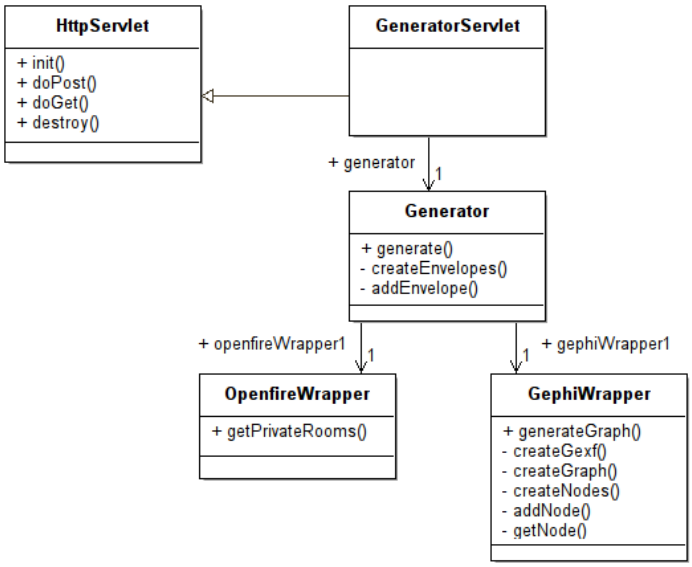

Figure 3.3: Generator Servlet class diagram

Openfire Wrapper

The Openfire wrapper object has a simple role in this scenario. It retrieves all private rooms from the Openfire server and returns them to the Generator object.

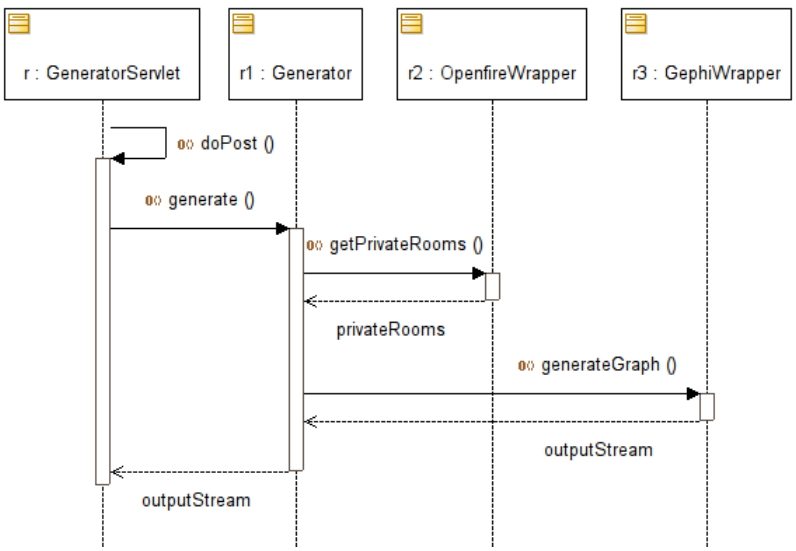

Figure 3.4: Generator Servlet sequence diagram

Gephi Wrapper

The Gephi Wrapper is a little bit more complex. It receives a list of envelopes from the Generator object and the output stream. From the list of envelopes, it creates a set of nodes[1]. Each node represents an individual user. The set has no duplicates. Afterwards, it opens each envelope and creates a linkage between nodes based on the from and to parameters. Then it sets the weight parameter of each linkage based on the count parameter of envelope. At the end of this process, the Gephi Wrapper generates a graph into the output stream.

3.3 Liferay Chat Plugin

The Liferay Chat Plugin consists of the backend and frontend side. The backend runs in a servlet container on the server. The frontend is in a form of javascript code, which runs in the user's web browser.

3.3.1 Backend

The backend copies the class structure from the analysis and design chapter (Figures 3.5 and 3.6). The Openfire connection is accessible via the Connection class which is a part of the Smack library.

The Jabber Implementation class holds a connection object for each user who has logged in. Whenever the user logs out, the Connection object is destroyed. The Chat Portlet class extends an MVCPortlet [17] from the Spring Framework [18] library, which is already linked, to Liferay. The Chat Poller Processor extends the Base Poller Processor, which is an integral part of Liferay.

Moreover, the Jabber Implementation class holds a Conversation Container that contains all conversations related to the particular user. Objects which can be distributed to the frontend (e.g. Buddy, Message, Conversation and Room) implement JSONable interface with the *toJSON()* method. Due to this approach, JSON mapping is handled within the object itself and thus does not need to be done in the controller classes.

[1] Each envelope has a from and to parameter which contains the username

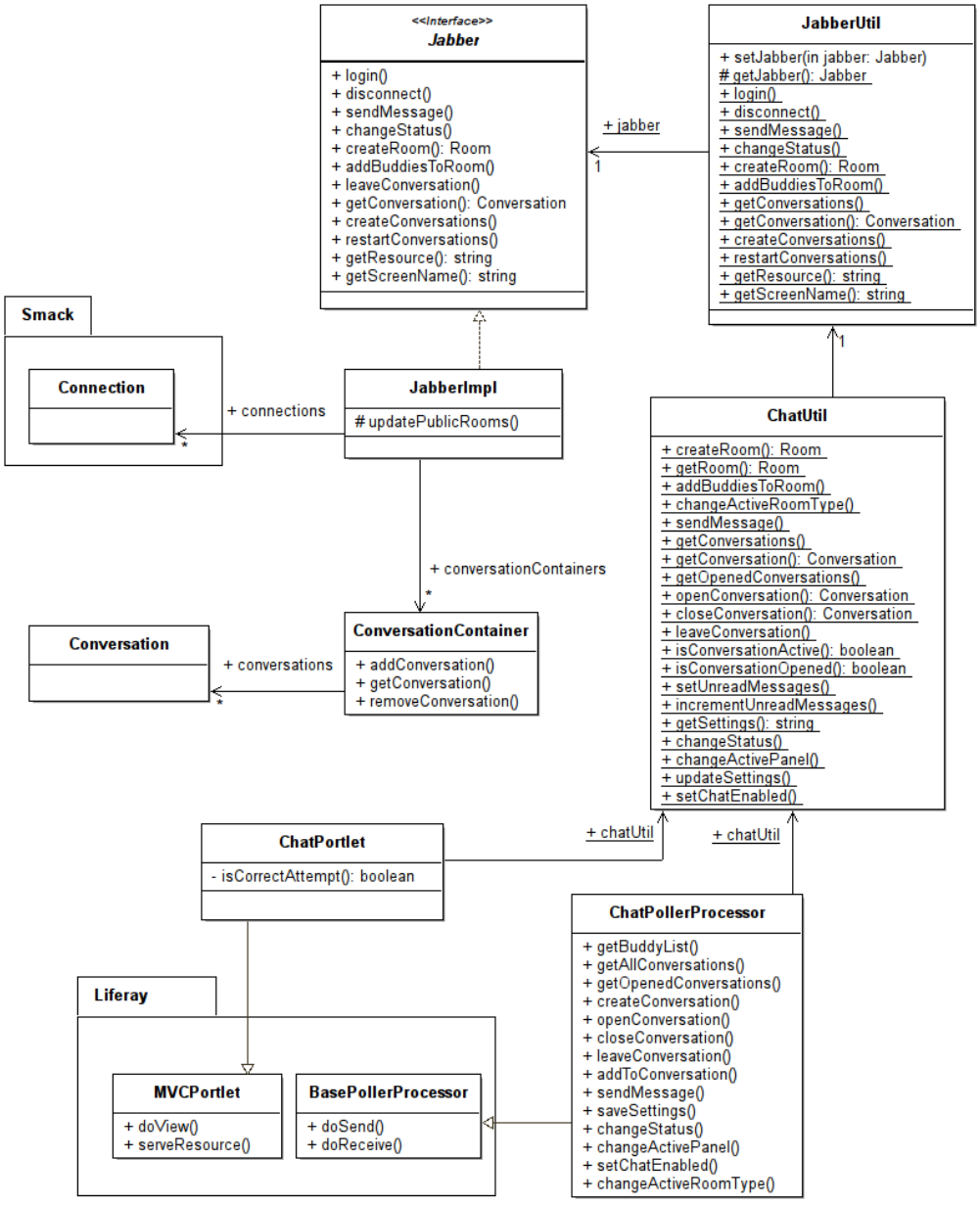

Figure 3.5: Backend class diagram

Some objects need to be stored in database (e.g. Room). Liferay offers the Service Builder [43] tool, which may be used for such purposes. It automates the creation of interfaces and classes that are used by a given portal or portlet. This includes code for EJBs, Spring, Persistence, and Model. Persistence classes can be stored in the database[2]. The input to the Service Builder is an XML file[3] located in */WEB-INF/service.xml*. Class generation can be run via the ant tool:

```
ant build-service
```

The Builder then generates several classes to the */WEB-INF/service* folder. Those files are not allowed to be changed by the programmer. The classes that can be modified are located in the */WEB-INF/src* folder under the **.service* package.

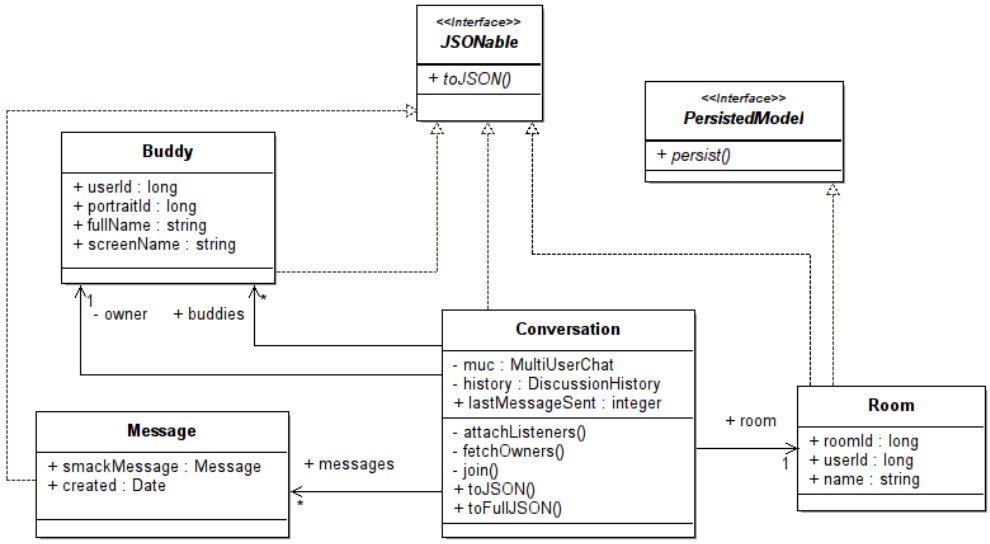

Figure 3.6: Conversation class diagram

[2]Specifically, classes that implement the PersistedModel interface

[3]The DTD for the XML file can be found here: http://www.liferay.com/dtd/liferay-service-builder_6_0_0.dtd

3.3.2 Frontend

The frontend mainly consists of javascript files located in the *docroot/js* folder. All chat classes and libraries are dynamically loaded in the *main.js* file[4]. After it loads all resources it initializes Liferay.Chat.Manager class by calling its *init()* method. The class diagram is shown on the Figure 3.7. All class names start with the Liferay.Chat prefix.

Alloy UI

The frontend fully exploits the Alloy UI framework functionality, which is built on the top of the YUI framework. The Alloy uses the concept of sandboxes. First, the programmer defines which packages he/she wants to use. Those packages are then used inside of the sandbox. This approach allows the code to run as leanly as possible, and load only what is needed. Furthermore, the sandbox pattern ensures that the custom code does not pollute the global scope of the page or interfere with other javascript libraries. The sandbox can be created by the following code:

```
AUI().use('aui-base', 'anim', function(A) {
// Custom code
});
```

AUI() creates a brand new AUI instance without any active modules. Those are then listed inside the *use()* function. The last parameter has to be a callback function. Parameter A, which is passed to the callback function, is called the "Global object". It stores all Alloy objects and classes. This concept is crucial because all Alloy elements, events and functions are called via this object.

[4]To avoid library inconsistence, it loads JQuery only if it is not already included by some other component.

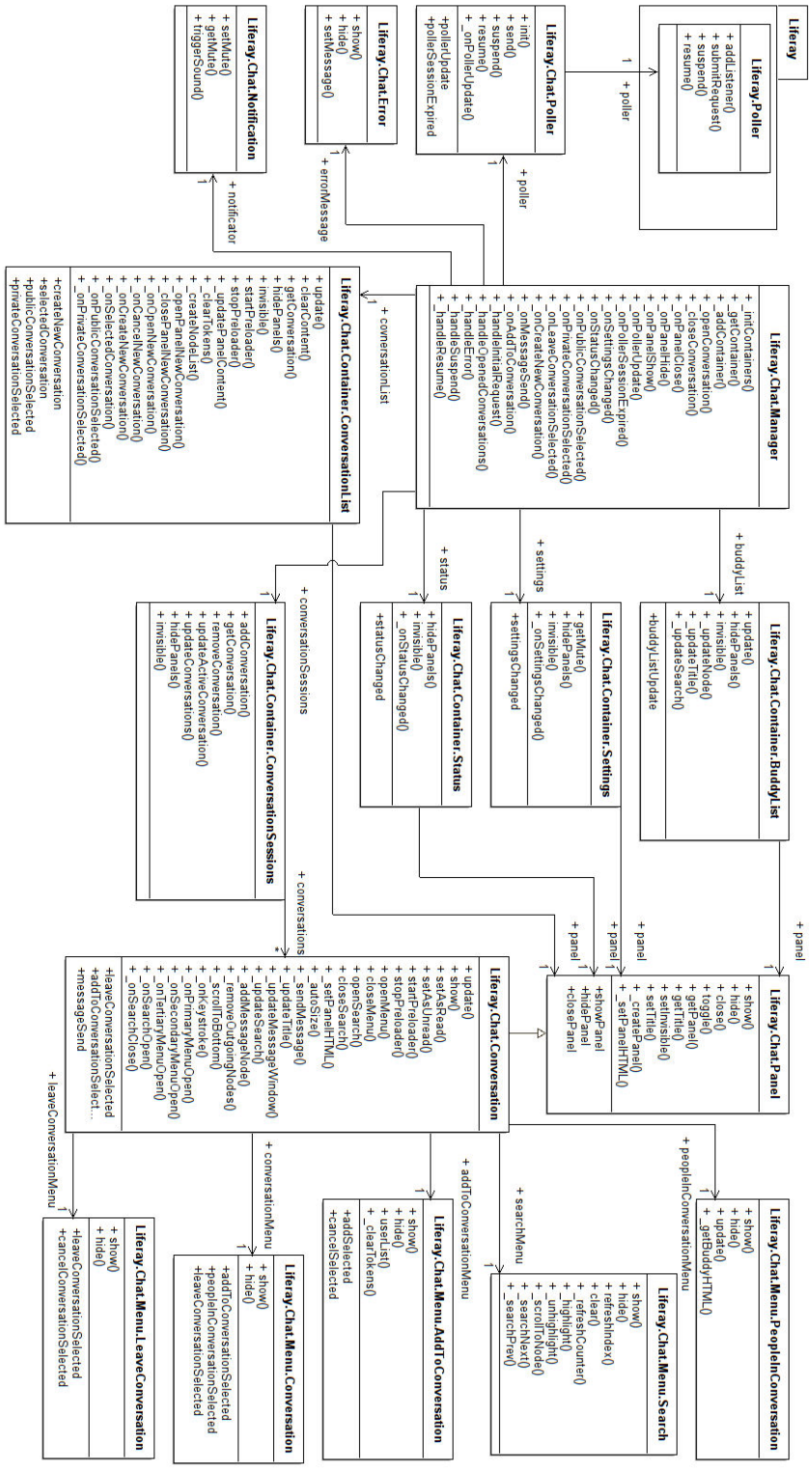

Figure 3.7: Frontend class diagram

Chapter 4

Testing and Experiments

The project is currently in the testing phase. The following chapter describes tests and experiments that have already been performed. Moreover, it includes issues, which have already been localized.

4.1 Unit testing

The unit testing is a popular way to test the source code. Unfortunately, Liferay Chat Portlet, whose source code was used as a basis for the Liferay Chat Plugin, does not support them. Therefore, they had to be implemented later[1].

The unit tests are mostly used to test a simple application logic, typically single methods [46]. However, the Liferay Chat Plugin does not contain many instances of such application logic. It usually communicates with multiple resources, which are located outside of the application environment container (e.g. Openfire server or the frontend).

On the other hand, the architecture of the Liferay Chat Portlet consists of nicely separated layers that can be easily replaced by the testing layers (Figure 4.1). For example JabberUtil class communicates with Openfire via the JabberImpl class, which implements the Jabber interface. Due to this approach, it can be easily replaced with any other class that implements the same interface. In our case, it is the JabberTestImpl class. The given class simulates the behaviour of a real Jabber class but does not communicate with the actual Jabber server. As a result, the programmer may run unit tests on the particular component without the need of the Openfire server.

[1] The unit test are usually created at the beginning of the implementation process.

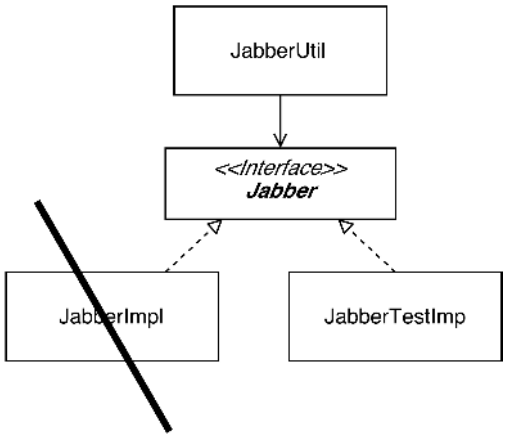

Figure 4.1: JabberImpl class replaced with JabberTestImpl class

Currently, such layers are being developed. Due to them, it will be possible to cover as much of the source code with the unit tests as desired.

4.2 Testing environment

The project is currently being tested in the faculty testing server infrastructure (Figure 4.2).

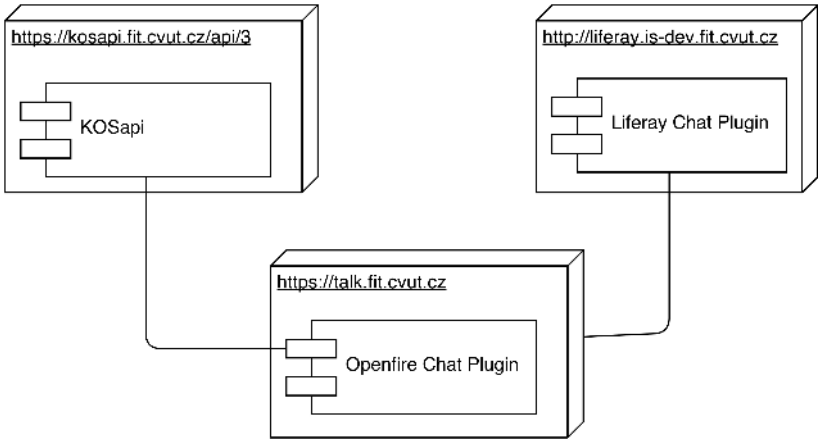

Figure 4.2: JabberImpl class replaced with JabberTestImpl class

KOSapi is already in the production phase and accessible at the following URL:

`https://kosapi.fit.cvut.cz/api/3`

Openfire server is fully functional and running at:

`https://talk.fit.cvut.cz`

The admin console of the Openfire server can be accessed at the same url but via the 9091 port. Liferay is currently deployed on the development server at the following URL:

`http://liferay.is-dev.fit.cvut.cz`

After the testing phase, the Liferay Chat Plugin will be deployed to the stage server and later on to the production server at:

`http://is.fit.cvut.cz`

4.3 Deployment

During the testing phase plugins need to be deployed quite often. The following text describes this process.

4.3.1 Liferay Chat Plugin

The plugin needs to be built before deployment. The Liferay Chat Portlet is a part of the liferay-plugins repository [12]. The built process is thus based on the development guide, which is located within the repository[2]. Before you start the building process follow these steps:

For demonstration purposes, let's pretend your user name is *joe* and you have a Liferay instance bundled with Apache Tomcat running in your */home/joe/* directory.

[2]Liferay Chat Plugin is currently build via the Ant tool. The build process is thus quite complicated. However, Ant will be replaced with Maven as a part of the future improvements.

1. Fork the liferay-plugins repository [12].
2. Clone your fork of the repository.
3. Create a *textbuild.[username].properties* file in the root directory of your liferay-plugins repository clone. Be sure to replace *[username]* with your user name. The file path should be similar to:

   `/home/joe/liferay-plugins/build.joe.properties`
4. In your *build.[username].properties* file, specify the `app.server.dir` property set to the path of your app server:

   `app.server.dir =`

   `/home/joe/liferay-portal-6.1.1-ga2/tomcat-7.0.27`
5. Replace the */home/joe/liferay-plugins/portlets/chat-portlet* folder with the similar folder from the enclosed DVD.
6. Edit the *chat-portlet/docroot/WEB-INF/src/portlet-ext.properties* file. The following lines show the example which works within the testing environment which was described before:

   ```
   jabber.host=talk.fit.cvut.cz
   jabber.port=5222
   jabber.service.name=talk.fit.cvut.cz
   jabber.service.multichat.name=conference.talk.fit.cvut.cz
   jabber.resource=Liferay
   jabber.sock5.proxy.enabled=false
   jabber.sock5.proxy.port=-1
   jabber.prefix.private=private_
   jabber.prefix.public=public_
   ```
7. Modify the *webapps/ROOT/WEB-INF/classes/portal-ext.properties* file which is located in the tomcat directory. The file suppose to define the `portlet.add.default.resource.check.whitelist` property which should contain the chat portlet identifier:

   `1_WAR_chatportlet`

   If there is no such property or if it does not contain the identifier copy the following line into the file:

   ```
   portlet.add.default.resource.check.whitelist=3,
   56_INSTANCE_0000,58,82,86,87,88,103,113,145,164,166,170,
   177,1_WAR_chatportlet
   ```

8. Navigate to the directory of a plugin and deploy it using Ant:
   ```
   cd /home/joe/liferay-plugins/portlets/chat-portlet
   ant deploy
   ```

The plugin compiles, its WAR file is built to the plugin's dist directory.

To deploy the WAR file within the testing environment follow the steps:

1. Copy the file from your local drive to the development server:
   ```
   scp -P 2210 chat-portlet-6.1.0.1.war
   username@is.fit.cvut.cz:
   ```

2. Login to the server:
   ```
   ssh -p 2210 username@is.fit.cvut.cz
   ```

3. Copy the file to the Liferay's deploy folder:
   ```
   sudo -u liferay cp  chat-portlet-6.1.0.1.war
   /opt/liferay/deploy
   ```

4.3.2 Openfire plugins

According to the Openfire Plugin Developer Guide, all Openfire plugins must be deployed as JAR or WAR files. When a JAR or WAR is not present for the plugin, Openfire assumes that the file has been deleted and that the user wants to destroy the plugin, so it also deletes the directory. To build the JAR file follow the steps:

1. Download the latest Openfire source code [26].

2. Remove all unnecessary plugins from the src/plugins directory except of the admin plugin.

3. Go to the openfire folder and build the openfire server:
   ```
   ant openfire
   ```

4. Copy the chat plugin and the data mining plugin source code from the enclosed DVD into the src/plugins directory.

5. Copy the modified build file from the enclosed DVD into the build folder.

6. Build plugins:
   ```
   ant plugins
   ```

The plugins compile and their JAR files are built to the target/openfire/plugins directory. The given files can be uploaded to the Openfire via the admin console as described in the user guide.

4.4 Known issues

This chapter discusses the issues that have appeared during the testing phase.

Message thread does not load

Problem: This problem occurs accidentally. Sometimes if the user jumps to a different page, and the conversation panel is open, the message feed does not load.

Possible solution: It probably has something to do with the chat poller concept. Because of the performance issues, the message feed is loaded at the initial response. All other responses contain newly added messages only. If the initial response fails, then the message feed does not appear within the conversation panel.

User is not logged out from the Openfire server properly

Problem: If the user closes the web browser window, his/her status is not changed to unavailable on the Openfire server side.

Possible solution: When users close the web browser window, his/her status is changed to *jabber.status.off* on the Liferay Chat Plugin backend side only. It should be changed on the Openfire server as well. On the other hand, Openfire has a timer which sets the status to unavailable after 2 minutes of user inactivity.

Possible memory leak

Problem: During the deployment process the following message appears:

```
The web application [/chat-portlet] appears to have started a
thread named [Smack Packet Reader (1)] but has failed to stop
it. This is very likely to create a memory leak.
```

Possible solution: The given problem is inherited from the Liferay Chat Portlet. It should be tested and repaired soon.

4.5 Experiments

During the testing phase, several experiments with the data mining plugin and the social network analysis were performed. For example, figure 4.3 shows the result of the community analysis[3] experiment. The GEXF file was generated via the Openfire Data Mining plugin and opened in Gephi. The graph has six nodes – students and several edges, which display the communication between them. Afterwards, the community analysis was performed. The resulting graph has shown two communities – red and blue.

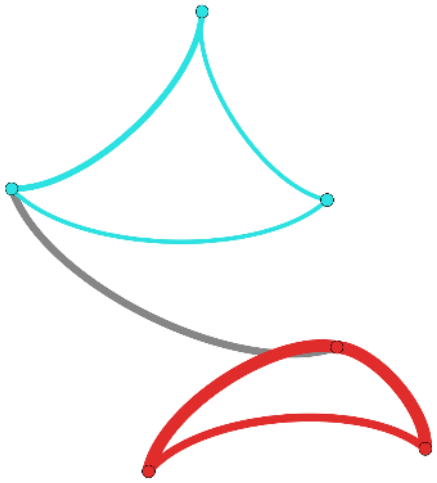

Figure 4.3: The community analysis experiment

The given results might, for example, identify groups of friends within the University based on the their frequent communication. The given results might be used during the composition of the student's schedule, e.g. the system might automatically put friends in the same class.

The traditional method for identifying communities in networks is hierarchical clustering: "Given a set of N nodes to be clustered, and an N x N distance (or similarity) matrix, the basic process of hierarchical clustering is this: Start by assigning each node its own cluster, so that if you have N nodes, you now have N clusters, each containing just one node. Let the distances between the clusters equal the distances between the nodes they contain. Find the closest (or most similar) pair of clusters and merge them into a single cluster, so that now you have one less cluster. Compute dis-

[3]The community analysis studies strong social ties within the network.

tances between the new cluster and each of the old clusters. Repeat until all nodes are clustered into a single cluster of size N" [3].

Figure Y. shows another example of a much bigger dataset [54] which contains many nodes, edges and thus more communities.

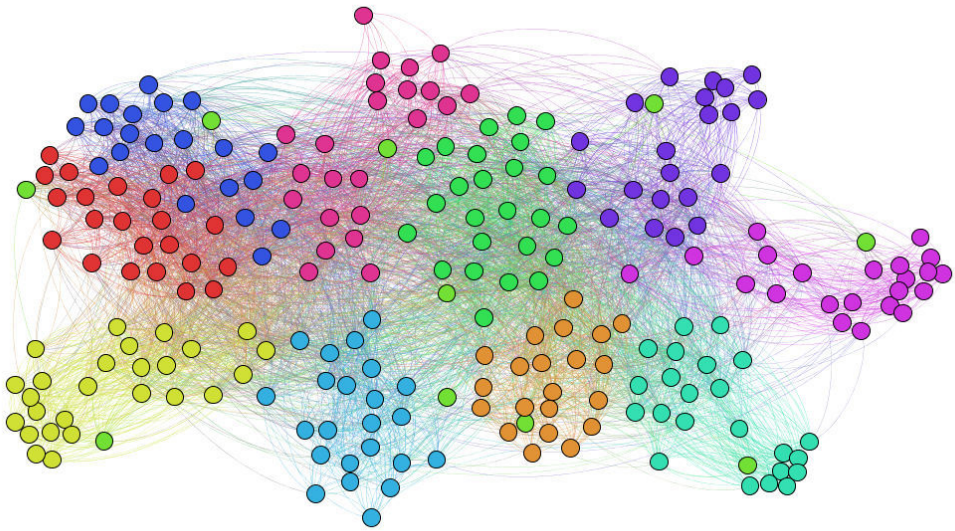

Figure 4.4: Large dataset with multiple communities

Conclusion

The requirements of this project were successfully met. The newly created communication system aims to be a frequently used channel that will connect multiple groups like students, teachers, employees and alumni.

The final product contains three independent plugins:

- Liferay Chat Plugin,
- Openfire Chat Plugin,
- Openfire Data Mining Plugin.

The given plugins form a modern instant messaging communication system in the environment of the Liferay portal. The Liferay Chat Plugin consists of a modern javascript interactive user interface and a backend server side, which is responsible for correct message delivery, conversation creation and communication with the Openfire server.

All conversations are stored on a separate, secured Openfire server. Therefore, it is possible to perform a social network analysis on the data located in the server database via the Open Chat Data Mining Plugin to show key social patterns that can be used for a variety of different research purposes. The given plugin generates a GEXF file that describes the connections between participants within private conversations.

The plugins were successfully tested in the emerging University Information System infrastructure. Therefore, they are ready for the production environment.

Future improvements

One-to-one conversations

Currently, all conversations are considered to be many-to-many, which makes it is easier to add new participants to a conversation. On the other hand, the XMPP protocol implements a different approach for either one-to-one or many-to-many conversations. This results in the fact that if the user wants to connect to the Openfire server with a different client than the Liferay Chat Plugin, he/she cannot chat with other participants directly. They must always create a multi- user chat room via Liferay first.

The next version of the Liferay Chat Plugin thus needs to implement both one-to-one and many-to-many conversations. Conversations which contains two participants will be handled as one-to-one. Conversations with more than two participants will be automatically considered as many-to-many. If users add new participants to the existing one-to-one conversation, it will be converted to a many-to-many format but not vice versa.

Open conversation by clicking on the user name or portrait

During the testing process it was discovered that users usually tend to create a new conversation by clicking on the buddy's username or portrait.

Therefore, the next version should implement this feature. If the user clicks on a buddy's username or portrait, the system will check if there is already a one-to-one conversation between those two participants. If so, it opens the given conversation. If not, it creates a new conversation. It should also be mentioned that the implementation of a one-to-one conversation is a prerequisite for this particular improvement.

Signalization of undelivered messages

Currently, there is no signalization of undelivered messages. Whenever the user sends a message that was not delivered to the server[4], the system should show a warning message. There should also be a resend option.

Signalization of an event when the user starts typing

There should be a signalization of an event when the user starts typing. The improvement cannot be implemented yet because the poller sends a

[4]Due to the e.g. connection problems

request every 4-8 seconds. The given time interval is simply too long. A possible solution would be to replace the Poller mechanism with the Comet model.

Comet model

The Poller mechanism, which is currently used, works on a ping-pong scheme. The frontend sends a request to the backend every 4-8 seconds (ping). The server responds immediately (pong). The ping interval cannot be shorter because server might be overloaded by a huge number of requests. On the other hand, the frontend cannot be notified about any change until it sends another request. The given downside might again be overcome by the comet model.

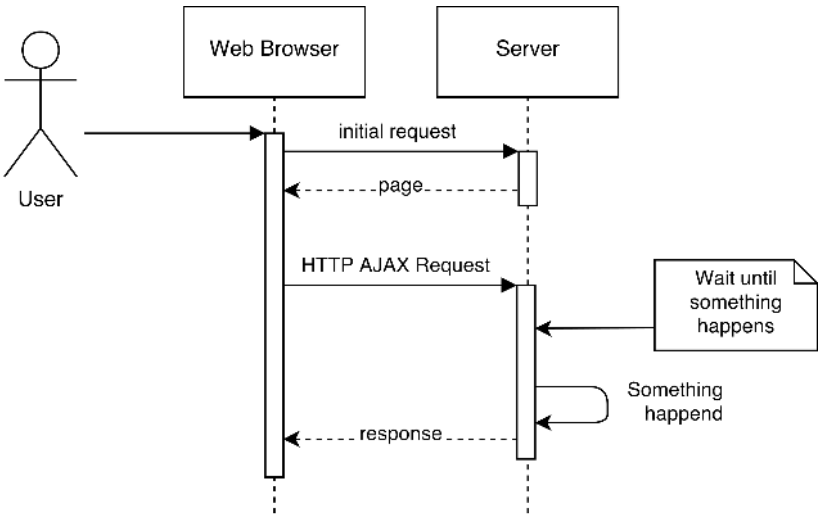

Figure 4.5: Comet AJAX mechanism

The comet mechanism is based on long-held HTTP requests (Figure 4.5). Due to this approach, the server can send data to the browser, without the browser explicitly requesting it. Comet has many implementations and techniques. Based on the requirements of this project, I would recommend using AJAX with a long polling approach. Although Liferay already supports comet integration [35], it is not well documented and thus needs more research.

Recurrent update of the message timestamp

The message timestamp (Figure 4.6) is updated on the full page refresh only. It would be appropriate to create a timer on the frontend side which would update the given timestamp every 30 seconds.

Figure 4.6: The message timestamp

References

[1] ALLOYUI: *AlloyUI 2.0.0pr5*. [software] [access 2013-05-07]. Available at WWW: <http://alloyui.com/>

[2] APPLE, INC.: *Apple (United Kingdom) – iPhone 5 - Learn about apps from the App Store*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <https://www.apple.com/uk/iphone/from-the-app-store/>

[3] ARENAS, A.; DANON, L.; D'IAZ-GUILERA, A.; etc.: Community analysis in social networks. *The European Physical Journal B*, volume 38, May 2004: pp. 373–380, doi:10.1371/journal.pone.0023176, ISSN 1434-6028

[4] FACEBOOK, INC.: *Facebook*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://facebook.com>

[5] FACEBOOK, INC.: *Facebook Chat*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <https://www.facebook.com/sitetour/chat.php>

[6] FICAROLA, Francesco: *gexf4j 0.4.2-BETA*. [software] [access 2013-05-07]. Available at WWW: <https://github.com/francesco-ficarola/gexf4j>

[7] FICAROLA, Francesco: *Gexf4j, a new Java library to create GEXF files — Gephi*. [online] c2008-2012, last revision 21st of May 2012 [cit. 2013-05-07]. Available at WWW: <https://gephi.org/2012/gexf4j-a-new-java-library-to-handle-gexf-file-format/>

[8] GEPHI CONSORTIUM: *Gephi*. [online] c2008-2012 [cit. 2013-05-07]. Available at WWW: <http://gephi.org>

[9] GEPHI CONSORTIUM: *Gephi 0.8.2-beta*. [software] [access 2013-05-07] User requirements: 500 MHz CPU, 128 MB RAM, OpenGL 1.2. Available at WWW: <https://gephi.org/users/download/>

[10] GEPHI CONSORTIUM: *Supported Graph Formats — Gephi, open source graph visualization software*. [online] c2008-2012 [cit. 2013-05-07]. Available at WWW: <https://gephi.org/users/supported-graph-formats/>

[11] GITHUB, INC.: *liferay (Liferay Inc.) · GitHub*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <https://github.com/liferay>

[12] GITHUB, INC.: *liferay/liferay-plugins · GitHub*. [online] c2013, last revision 9th of May 2013 [cit. 2013-05-07]. Available at WWW: <https://github.com/liferay/liferay-plugins>

[13] GOOGLE, INC.: *Android Apps on Google Play*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <https://play.google.com/store>

[14] GOOGLE, INC.: *Google +*. [online] [cit. 2013-05-07]. Available at WWW: <https://plus.google.com/>

[15] GOOGLE, INC.: *Google Talk - About*. [online] c2011 [cit. 2013-05-07]. Available at WWW: <http://www.google.com/talk/about.html>

[16] GOOGLE, INC.: *Video chat with up to nine friends - Google+*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.google.com/+/learnmore/hangouts>

[17] GOPIVOTAL, INC.: *Chapter 16. Portlet MVC Framework*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://static.springsource.org/spring/docs/2.0.8/reference/portlet.html>

[18] GOPIVOTAL, INC.: *SpringSource.org*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.springsource.org/>

[19] HANSEND, D.; SHNEIDERMAN, B.; SMITH, M.: *Analyzing social media networks with NodeXL: insights from a connected world*. Burlington: Morgan Kaufmann, first edition, 2011, 284 pp., ISBN 978-0-12-382229-1

[20] HEYMANN, S.; BASTIAN, M.: *GEXF File Format*. GEXF Working Group, [online] c2009 [cit. 2013-05-07]. Available at WWW: <http://gexf.net/format>

[21] IETF: *Internet Engineering Task Force (IETF)*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.ietf.org>

[22] JIRŮTKA, Jakub: *Main - KOSapi - Projekt KOSapi*. [online] last revision 29th of May 2013 [cit. 2013-05-07]. Available at WWW: <https://kosapi.fit.cvut.cz/projects/kosapi/wiki>

[23] JIVE SOFTWARE: *Ignite Realtime: About*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/about/index.jsp>

[24] JIVE SOFTWARE: *Ignite Realtime: Openfire Server*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/projects/openfire/index.jsp>

[25] JIVE SOFTWARE: *Ignite Realtime: Smack API*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/projects/smack/index.jsp>

[26] JIVE SOFTWARE: *Ignite Realtime: Source Code*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/downloads/source.jsp>

[27] JIVE SOFTWARE: *Openfire 3.8.1*. [software] [access 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/downloads/index.jsp>

[28] JIVE SOFTWARE: *Openfire: Installation Guide*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/install-guide.html>

[29] JIVE SOFTWARE: *Openfire: Plugin Developer Guide*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/plugin-dev-guide.html>

[30] JIVE SOFTWARE: *Openfire: Protocol Support*. [online] [cit. 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/protocol-support.html>

[31] JIVE SOFTWARE: *Smack 3.3.0*. [software] [access 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/downloads/index.jsp>

[32] KRAUT, R.; BRYNIN, M.; KIESLER, S.: *Computers, Phones, and the Internet: Domesticating Information Technology*. USA: Oxford University Press, first edition, 2006, 344 pp., ISBN 978-0-195-17963-7

[33] LIFERAY, INC.: *About Portlets - Wiki - Liferay.com*. [online] c2013, last revision 9th of October 2011 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/community/wiki/-/wiki/Main/About+Portlets>

[34] LIFERAY, INC.: *Chat Improvements - Wiki - Liferay.com*. [online] c2013, last revision 14th of February 2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/community/wiki/-/wiki/Proposals/Chat+Improvements>

[35] LIFERAY, INC.: *Comet Integration - Wiki - Liferay.com*. [online] c2013, last revision 6th of June 2011 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/community/wiki/-/wiki/1071674/Comet+Integration>

[36] LIFERAY, INC.: *Enterprise open source portal and collaboration software. - Liferay.com*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com>

[37] LIFERAY, INC.: *Hooks - Development - Liferay.com*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/hooks>

[38] LIFERAY, INC.: *Liferay Portal 6.1 Community Edition GA2*. [software] [access 2013-05-07]. Available at WWW: <http://www.igniterealtime.org/downloads/index.jsp>

[39] LIFERAY, INC.: *Marketplace - Liferay.com*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/marketplace>

[40] LIFERAY, INC.: *Performing a Custom Action - Development - Liferay.com*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/performing-a-custom-action>

[41] LIFERAY, INC.: *Plugin Management - User Guide - Liferay.com*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/documentation/liferay-portal/6.1/user-guide/-/ai/lp-6-1-ugen15-plugin-management-0>

[42] LIFERAY, INC.: *Portal, Content, and Collaboration for the Enterprise. - Liferay.com*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/products/liferay-portal/overview>

[43] LIFERAY, INC.: *Service Builder - Wiki - Liferay.com*. [online] c2013, last revision 16th of June 2011 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/community/wiki/-/wiki/Main/Service+Builder>

[44] LIFERAY, INC.: *Staying in touch with the Chat - User Guide - Liferay.com*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://www.liferay.com/documentation/liferay-portal/6.1/user-guide/-/ai/ch-4>

[45] LIFERAY, INC.: *System Dashboard - Liferay Issues*. [online] [cit. 2013-05-07]. Available at WWW: <http://issues.liferay.com/>

[46] LORENZ, Jesse: *How to Write Good Unit Tests - developer.force.com*. [online] c2000-2013 [cit. 2013-05-07]. Available at WWW: <http://wiki.developerforce.com/page/How_to_Write_Good_Unit_Tests>

[47] LUNDGREN, E.; CAVANAUGH, N.: *AlloyUI*. ALLOY, [online] [cit. 2013-05-07]. Available at WWW: <http://alloyui.com>

[48] MAGSINO, Sammantha: *Applications of Social Network Analysis for Building Community Disaster Resilience : Workshop Summary*. Washington, DC, USA: National Academies Press, 2009, 82 pp., ISBN 0-309-14094-3

[49] OBJECT MANAGEMENT GROUP, INC.: *Object Management Group – UML*. [online] c1997-2013, last revision 15th of April 2013 [cit. 2013-05-07]. Available at WWW: <http://uml.org>

[50] ORACLE CORPORATION: *JSR-000286 Portlet Specification 2.0 - Final Release*. [online] c2013 [cit. 2013-05-07]. Available at WWW: <http://jcp.org/aboutJava/communityprocess/final/jsr286/>

[51] REID, N.; SMITH, B. W.: Social network analysis. *Economic Development Journal*, volume 8, no. 3, Summer 2009, pp. 48–55, ISSN 15391922.

[52] SAINT-ANDRE, P.: *XEP-0045: Multi-User Chat*. XMPP STANDARDS FOUNDATION, [online] c1999-2013, last revision 8th of February 2012 [cit. 2013-05-07]. Available at WWW: <http://xmpp.org/extensions/xep-0045.html>

[53] SEZOV, Rich: *Liferay in Action: The Official Guide to Liferay Portal Development*. Shelter Island, NY, USA: Manning Publications, first edition, 2012, 376 pp., ISBN 9781935182825

[54] STEHLÉ, J.; VIORIN, N.; BARRAT, A.; etc.: High-Resolution Measurements of Face-to-Face Contact Patterns in a Primary School. *PLOS ONE*, volume 6, no. 8, 08 2011: p. e23176, doi:10.1371/journal.pone.0023176, [online] [cit. 2013-05-07]. Available at WWW: <http://dx.doi.org/10.1371/journal.pone.0023176>

[55] XMPP STANDARDS FOUNDATION: *Jabber Resources - XMPP Wiki*. [online] last revision 23rd of February 2010 [cit. 2013-05-07]. Available at WWW: <http://wiki.xmpp.org/web/Jabber_Resources>

[56] XMPP STANDARDS FOUNDATION: *The XMPP Standards Foundation*. [online] [cit. 2013-05-07]. Available at WWW: <http://xmpp.org>

[57] YAHOO, INC.: *YUI Global Object - YUI Library*. [online] c2006-2013 [cit. 2013-05-07]. Available at WWW: <http://yuilibrary.com/yui/docs/yui>

[58] YAHOO, INC.: *YUI Library*. [online] c2006-2013 [cit. 2013-05-07]. Available at WWW: <http://yuilibrary.com/>

Appendix A

Acronyms

AJAX Asynchronous JavaScript and XML

ABNF Augmented Backus–Naur Form

AOL America Online

API Application Programming Interface

CDDL Common Development and Distribution License

CSS Cascading Style Sheets

CSV Comma-separated Values

DTD Document Type Definition

EJB Enterprise Java Beans

GDF Geographic Data Files

GEXF Graph Exchange XML Format

GML Graph Modelling Language

GPL General Public License

HSQL Hyper Structured Query Language

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

ICQ I Seek You

IETF Internet Engineering Task Force

IM Instant Messaging

JAR Java Archive

JID Jabber Identificator

JSON JavaScript Object Notation

JSP Java Server Pages

KOS Komponenta Student

LDAP Lightweight Directory Access Protocol

LGPL Lesser General Public License

MVC Model View Controller

REST Representational State Transfer

RTC Real Time Communication

SNA Social Network Analysis

SSL Secure Sockets Layer

TLS Transport Layer Security

UI User Interface

UML Unified Modeling Language

URL Uniform Resource Locator

WAR Web Archive

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

YUI Yahoo! User Interface

Appendix B

User guide

B.1 Liferay Chat Plugin

The Chat interface is visible at the bottom right part of the screen B.1. It is divided into the static and dynamic part. The static part contains a Conversations List, Settings, Buddy List and Status Panel, which will always be visible[1]. The dynamic part might be hidden[2] or may contain any number of conversations.

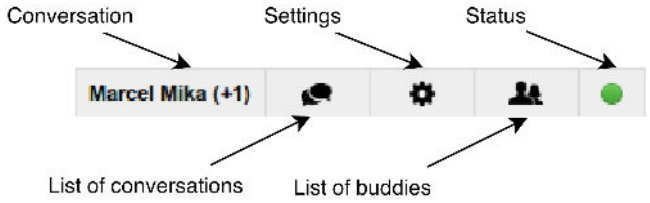

Figure B.1: Chat interface

Open panel: Click on the panel icon or title.

Minimize panel: There are several options to minimize the panel:

- Click on the minimize button at the top right part of the panel.
- Click on any other panel icon or text[3].
- Click on any currently open panel icon or text.

[1] Although, If the chat is turned off they will be hidden.

[2] If there are no opened conversations.

[3] There might be only one panel opened at the same time. If you click on any other panel, the current panel will be minimized.

Close panel: Click on the close button at the top right part of the panel.

B.1.1 Status

Status shows your current level of availability. You can be either:
- *Online* - available, ready for chat.
- *Busy* - busy, but still interested in chatting.
- *Unavailable* - available in the urgent situations, not interested in chatting
- *Off* - you are not logged in or you turned the chat off

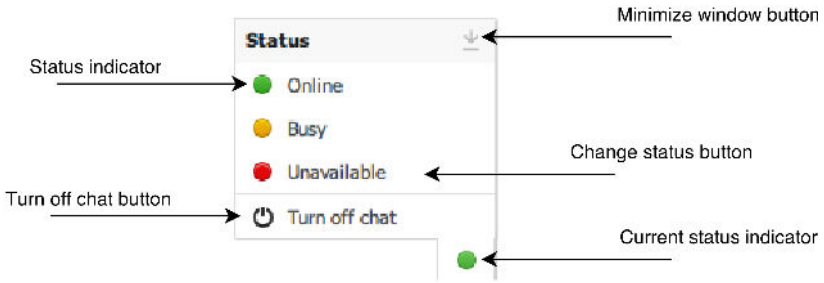

Figure B.2: Status panel

What is my current status? Your current status is displayed as a circular icon at the rightmost part of the chat interface. It can be distinguished by a color[4] or by a tooltip which is displayed whenever you point a cursor above the circular icon.

Change status:

1. Open the status panel (Figure B.2).
2. Click on the status you want to change.
3. Current status indicator will be changed based on the status you have selected.

Turn the chat off:

1. Open the status panel.
2. Choose "Turn off chat" option.
3. All panels except the status panel will be closed and the chat will be turned off.

[4]green – available; yellow – bussy; red – unavailable

Turn the chat on:

1. Open the status panel.
2. Select one of the statuses (e.g. Online, Busy, Unavailable).

B.1.2 Buddy list

The buddy list (Figure B.3) aggregates all participants who are in the same conversations as the user. Each item on the list contains the user's avatar, full user name and current status indicator. Users can be filtered via the search box.

Filter buddies:

1. Click on the search box.
2. Start typing the buddy's name.
3. The list will be filtered based on the search phrase.

Remove filter Clear search phrase.

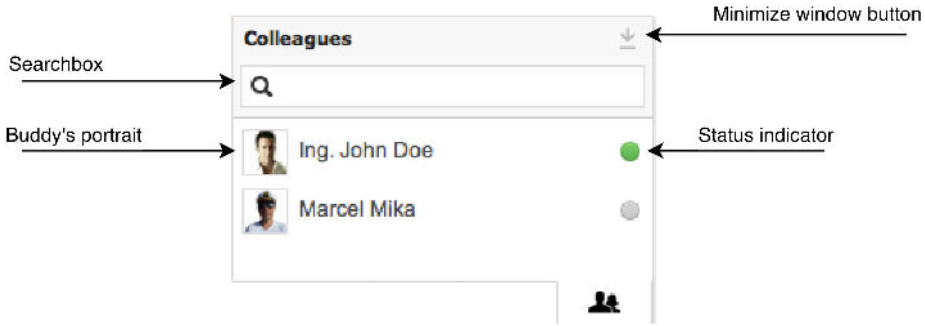

Figure B.3: Buddy list panel

B.1.3 Settings

Whenever you receive a new message in a conversation, which is not open, the system plays a notification sound. This option can be turned on/off via the settings panel (Figure B.4).

Turn the notification sound on/off: Open settings panel. Tick the play sound box.

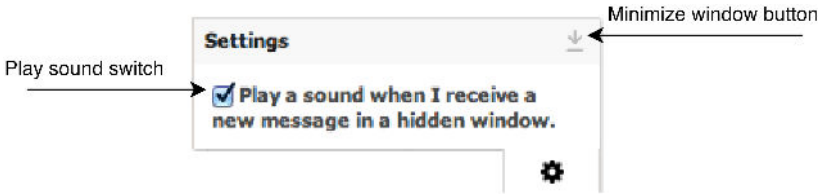

Figure B.4: Settings panel

B.1.4 Conversation List

The conversation list (Figure B.5) shows all conversations the user is participating in. Conversations might be either private or public.

Private conversation: The conversation created by the user or created by somebody who listed the user as a participant.

Public conversation: Automatically generated conversation based on the group the user belongs to[5].

Open conversation: Click on the conversation within the list.

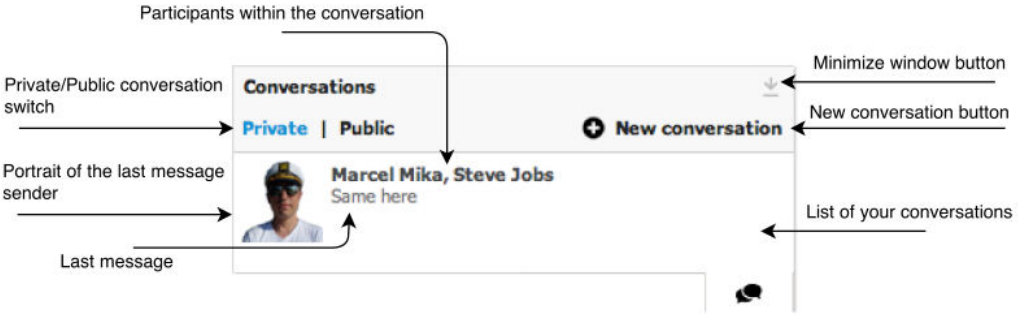

Figure B.5: Conversation list panel

Create new conversation:

1. Click on the "New conversation button". A new conversation menu should appear (Figure B.6). The menu contains a list of participants and a message box.
2. Click on the list of participants and start typing the buddy's name you want to add as a participant. A dropdown list of available buddies should appear.
3. Choose one of the options.

[5]Currently there are two groups: alumni bachelor and alumni master.

4. If you want to remove participant click on the "Remove from the list of participants button".
5. If you want to add more participants, simply repeat step 2.
6. Write the initial message into the message box.
7. If you do not want to create new conversation click on the cancel button.
8. If you want to create new conversation click on the send button[6].

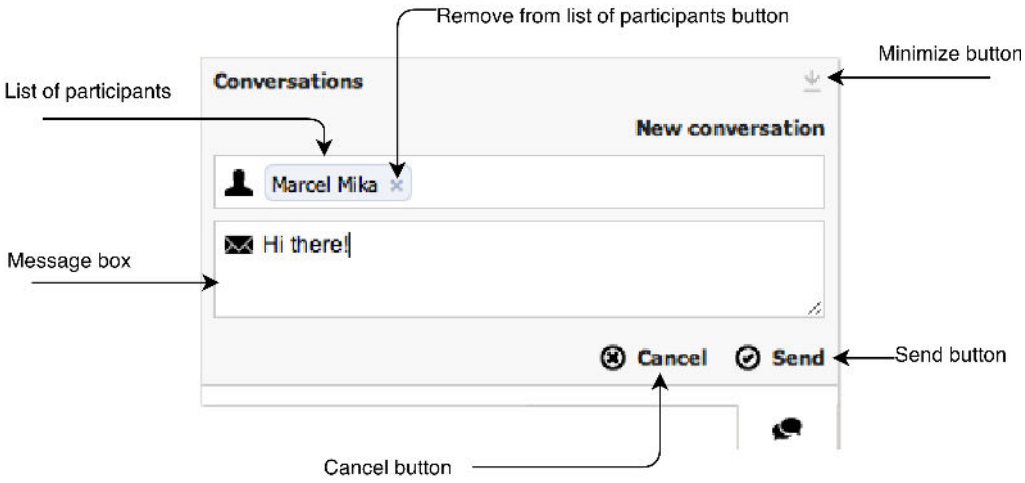

Figure B.6: New Conversation menu

B.1.5 Conversation

The user may open as many conversations as desired (Figure B.7). To distinguish between open panels, take a look at the panel title. The panel title contains the full name of the last message sender and the sum of other participants within the conversation.

Message feed: The message feed contains a list of all messages within the conversation. Each item in the list contains a buddy's portrait, name, message and the message timestamp.

[6]List of participants and message box cannot be empty. A new conversation will not be created until the user chooses at least one participant and writes at least one letter in the message box.

Send message:

1. Click in the message box.
2. Type your message.
3. Press the Enter key.

The message will be added to the message feed.

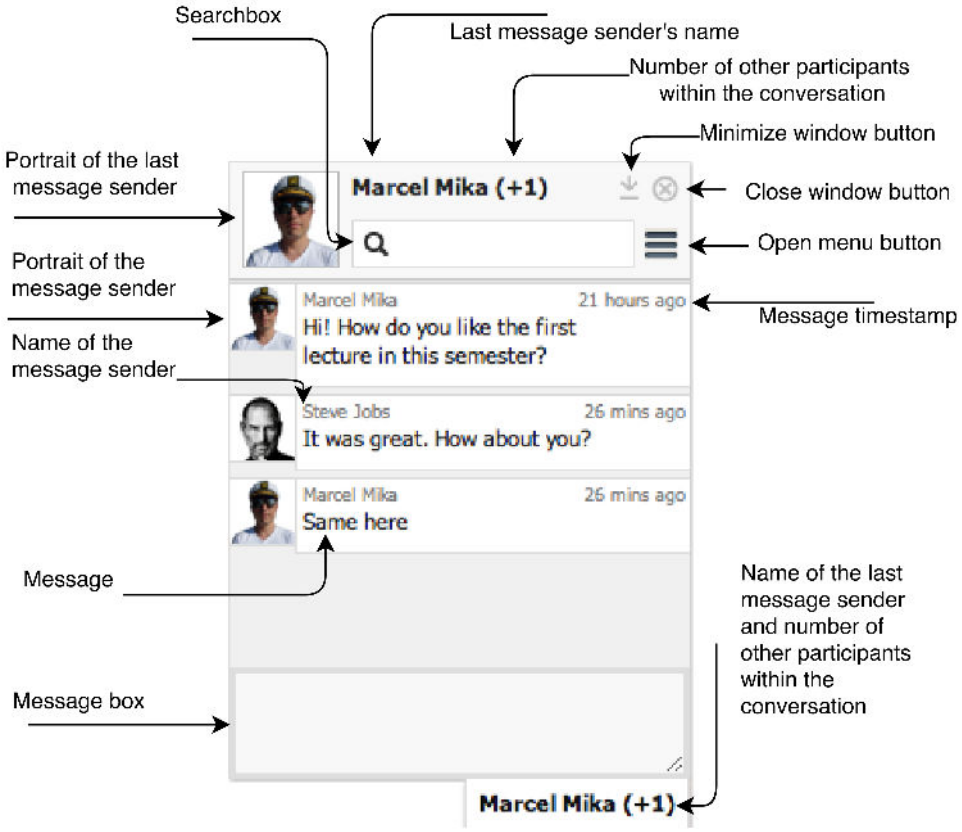

Figure B.7: Conversation panel

Search in messages:

1. Click into the search box (Figure B.8).
2. Type the word or phrase needed.
3. The message feed will automatically scroll to the first occurrence of the phrase and show the number of total results.
4. Multiple results can be viewed by clicking on the "Next" or "Previous" button.
5. To close search menu, click on the "Close search" button.

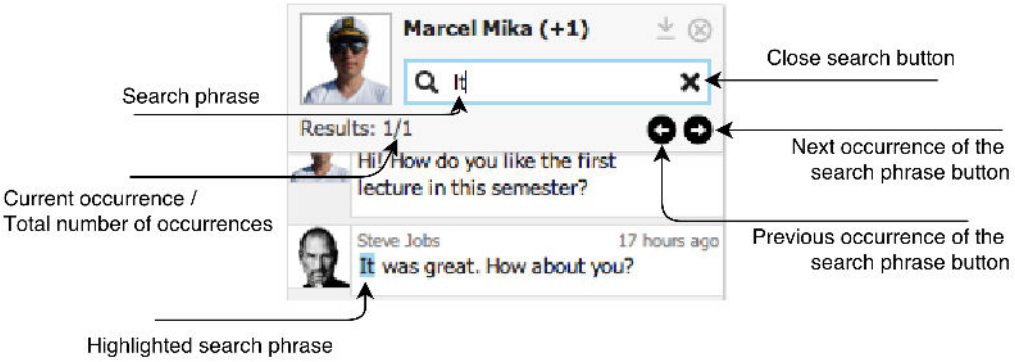

Figure B.8: Search menu

Open menu: Several menu options related to the opened conversation can be selected by clicking on the menu button. This will result in opening a menu with related options (Figure B.9).

Figure B.9: Opened menu

Add participants to conversation: In order to add more participants to the existing conversation (Figure B.10), simply:

1. Click on the "Add to conversation" option from the menu.
2. Click on the list of participants and start typing the buddy's name to add as a participant. A dropdown list of available buddies should appear.
3. Choose one of the options.
4. To remove a participant click on the "Remove from the list of participants" button.
5. To add more participants, simply repeat step 2.

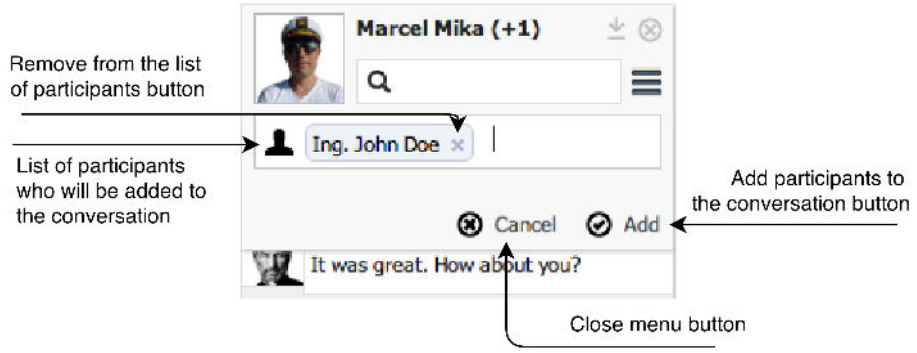

Figure B.10: Add Participants to Conversation menu

List of participants: To see a list of participants (Figure B.11) click on the "People in conversation" option in the menu.

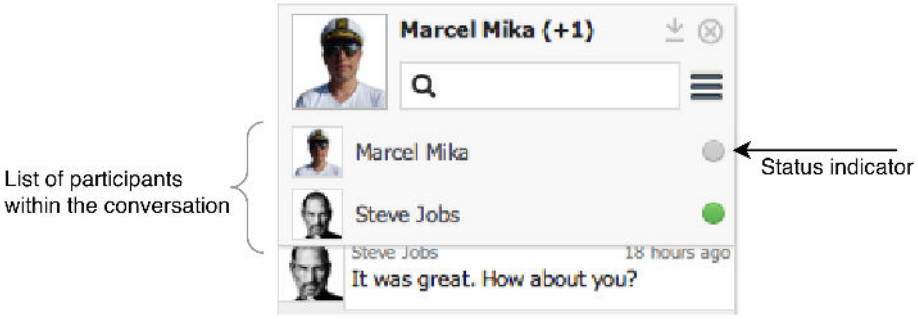

Figure B.11: List of Participants menu

Leave conversation: In order to exit the conversation (Figure B.12):

1. Choose "Leave conversation" option from the menu.
2. Click on the "Leave" button.
3. To remain in the conversation click on the "Cancel" button.

Figure B.12: Leave Conversation menu

B.2 Openfire Chat Plugin

To put the Openfire Chat Plugin into operation, an instance of the Openfire server must be installed and ran. Please consult the installation guide [28] for further information.

Installation:

1. Log into the Openfire admin console.
2. Click on the "Plugins" option from the primary navigation.
3. Click on the "Choose File" button.
4. Locate and choose *openfire-chat-plugin.jar* file.
5. Click on the "Upload Plugin" button.

The plugin should appear in the list of uploaded plugins B.13.

Change settings:

1. Log into the Openfire admin console.
2. Click on the "Server" option from the primary navigation.
3. Choose the "Server Settings" option from the secondary navigation.
4. Click on the "Chat Plugin" from the left side bar.
5. Chat Plugin Properties page will appear (Figure B.14)
6. Now, you can change the following settings:
    - *Global Properties* – includes properties, which are common for the whole Openfire Chat Plugin. The given properties should match the properties on the Liferay Chat Plugin side.
        - *Service name* – multi-user chat service name;
        - *Public room prefix* – prefix for the public room JID.

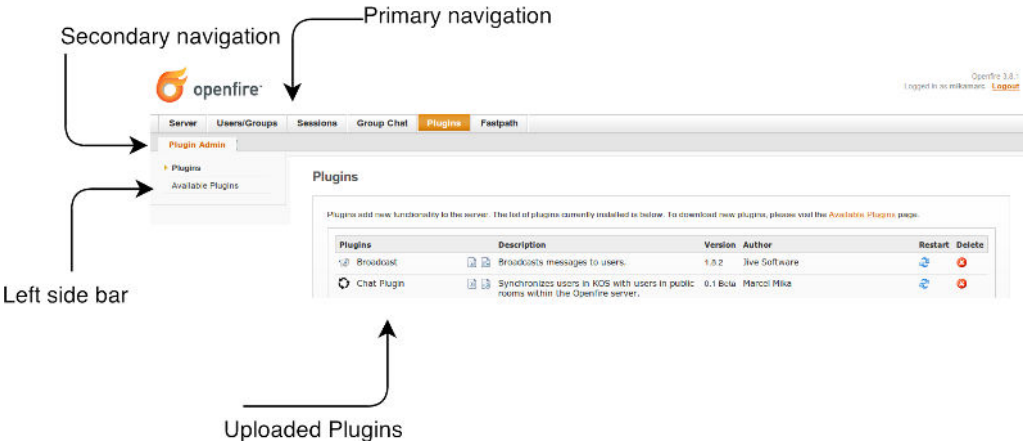

Figure B.13: The list of uploaded plugins

- *KOS Settings* – the properties related to the KOSapi service:
  - *KOSapi URL* – unique resource locator of the KOSapi REST service;
  - *Username* – username which is going to be used to login to the KOSapi REST service;
  - *Password* – password which is going to be used to login to the KOSapi REST service.
- *Public Rooms* – the settings related to the public room synchronization:
  - *Alumni Bachelor* – name of the Alumni Bachelor public room;
  - *Alumni Master* – name of the Alumni Master public room.

7. After you changed the settings you wanted click on the Save Settings button.

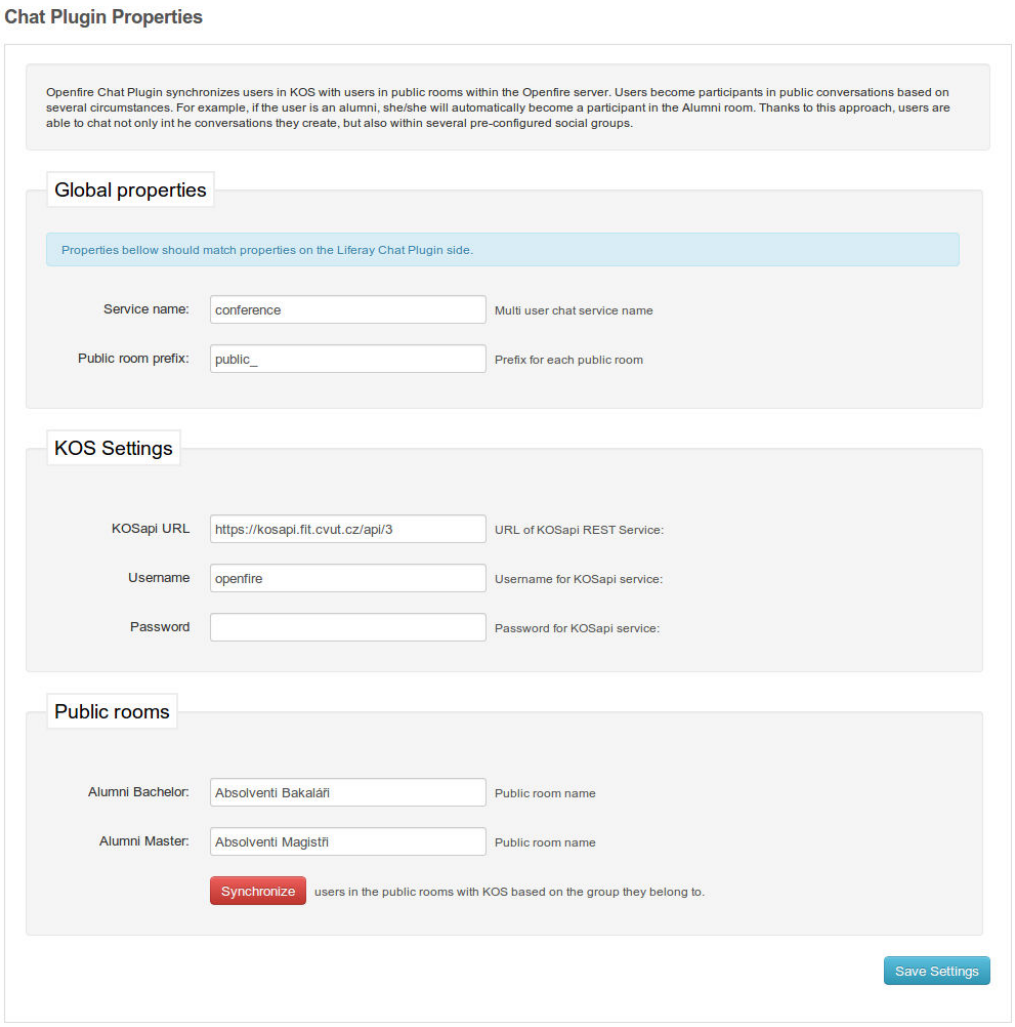

Figure B.14: Openfire Chat Plugin Properties

Create public rooms based on the data from KOS:

1. Log into the Openfire admin console.
2. Click on the "Server" option from the primary navigation.
3. Choose "Server Settings" option from the secondary navigation.
4. Click on the "Chat Plugin" from the left side bar.
5. Click on the "Synchronize" button within the Public Rooms section.

The system will log into the KOSapi service and download the list of alumni. Afterwards, the list will be used to create the particular public rooms and add the related users to them.

B.3 Openfire Data Mining Plugin

In order to put the Openfire Chat Plugin into operation, an instance of the Openfire server needs to be installed and ran. Please consult the installation guide [28] for further information.

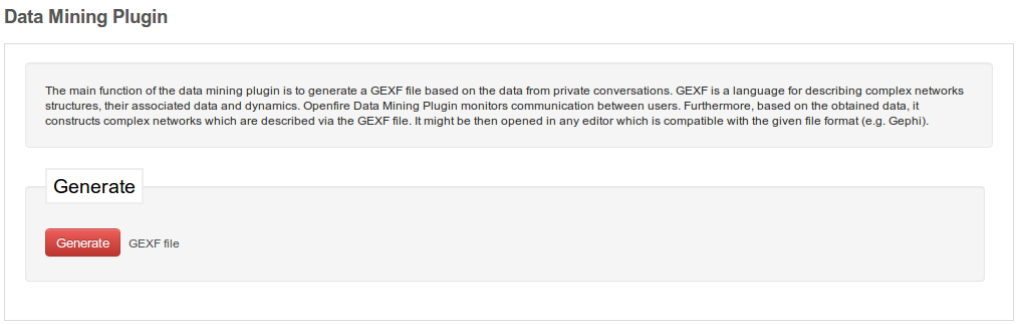

Figure B.15: Openfire Data Mining Plugin page

Generate GEXF file:

1. Log into the Openfire admin console.
2. Click on the "Server" option from the primary navigation.
3. Choose the "Server Settings" option from the secondary navigation.
4. Click on the "Data Mining Plugin" option from the left side bar.
5. Click on the "Generate" button within the Generate section (Figure B.15).

The system will generate a GEXF file. It may then be opened in any editor which is compatible with the given file format (e.g. Gephi).

Appendix C

Contents of enclosed DVD

- `readme.txt` ..... the file with DVD contents description
- `war` ..... the directory with Java web archives
- `src` ..... the directory of source codes
  - `liferay` ..... Liferay plugins implementation sources
    - `chat-portlet` ..... Liferay Chat Portlet implementation sources
  - `openfire` ..... Openfire plugins implementation sources
    - `chatPlugin` ..... Openfire Chat Plugin impl. sources
    - `dataMiningPlugin` .. Openfire Data Mining Plugin impl. sources
    - `build.xml` ..... modified build script
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `Marcel-Mika-thesis-2013.pdf` ..... the thesis text in PDF format
  - `Marcel-Mika-thesis-2013.ps` ..... the thesis text in PS format