

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Department of Computer Science

Master's Thesis

Appstore for LibreSocial- A Repository based App-Management for P2P Framework

Maitri Modasia
Matriculation number: 6747719
E-Mail: maitrim@uni-paderborn.de

Paderborn, den April 9, 2015

submitted to
Prof. Dr. Christian Scheideler
Dr.-Ing. Kalman Graffi

Declaration

Lorem ipsum dolor cit.

Paderborn, den April 9, 2015

Maitri Modasia

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Requirement Specification . . . 2
  - 1.2.1 Repositories for Appstore . . . 2
  - 1.2.2 Apps in a Repository . . . 3
  - 1.2.3 Publisher View . . . 3
  - 1.2.4 Shared Repositories View . . . 4
  - 1.2.5 Installed Apps View . . . 4
- 1.3 Role definition for users of appstore . . . 5
  - 1.3.1 Role of a Publisher . . . 5
  - 1.3.2 Role of a User . . . 5
- 1.4 Organization . . . 5

2 Background **7**
- 2.1 What is LibreSocial? . . . 7
  - 2.1.1 LibreSocial- Features in a nutshell . . . 8
  - 2.1.2 GUI/Plugins . . . 8
  - 2.1.3 Framework . . . 12
  - 2.1.4 P2P Overlay . . . 13
- 2.2 Overview of OSGi Framework . . . 14
  - 2.2.1 OSGi structure . . . 15
  - 2.2.2 OSGi bundles as plugins to LibreSocial . . . 17

Appendix **19**

List of Figures **21**

Bibliography **23**

1 Introduction

The idea of P2P(peer-to-peer) networks is not new and has been receiving high demands due to its scalability, fault tolerance, decentralization and self-organizing nature. There are different applications available today which are based on P2P technology such as BitTorrent, Gnutella, Napster, Skype etc. P2P applications have established themselves in the area of file-sharing and provide an alternative to the traditional client-server computing. It is not doubt that P2P applications are booming because of its distributed storage and secure communication features. However, diving into the area of social networking, currently there are highly matured OSNs(Online Social Networks) such as Facebook, MySpace, Twitter, LinkedIn etc. Such social interactive platforms help their users to communicate online and share information with each other. OSNs let their users create their personal or professional profiles, upload photos and videos, messaging, video-chatting, sharing documents, creating groups etc. Currently most of the online social networks are operated by servers that are overloaded by huge amount of user data. Server based IT architecture provides the desired performance but incur very high operational costs. Also, the users have concern about their security and privacy since the user is not the only master of his shared information. Alternatively, large scale efficient networks implementing p2p based architecture also exist. In P2P applications, maintenance and operational costs are shifted to the users. An idea of P2P based OSNs can interest sufficient number of users who advocate secure communication and distributed storage. LibreSocial earlier known as 'LifeSocial.KOM' is one such application which is completely decentralized in nature and provides similar functionalities as popular online social networks like user profiles, friend lists, wall entries, user groups, E-learning etc. LibreSocial has a plugin-based layered architecture made up of several individual plugins. Different functionalities of social apps are implemented in the form of these plugins. For example, Login, profile, messaging, files, photos etc., are different plugins aggregated together which provide interfaces for other plugins and the graphical user interface. These plugins are hosted on a platform called FreePastry which is an overlay network for connecting different peers and efficient data routing. Additional plugins can be integrated without disturbing the existing plugins and thus the functionality of the application can be extended.

1.1 Motivation

LibreSocial is highly extendible in nature. The idea behind writing this master thesis is to develop an 'Appstore' for LibreSocial. An Appstore is an independent plugin extension for repository based app management to existing LibreSocial. LibreSocial has plugin based architecture and is composed of different plugins. Such an application is suitable of extending its functionality by additional plugins. Appstore is basically a repository based app-management for LibreSocial. It serves as a platform for creating an exclusive 'AppMarket' managed by the users of LibreSocial. Having an Appstore allows users to browse and search through the catalog, publish, download and install different apps. This way, LibreSocial can be enriched with one more functionality. The main idea is to provide the users an opportunity to create their own plugins or apps based on the P2P framework. These new plugins or apps can be shared via the Appstore. One can publish new apps, release new versions, edit or delete the meta-data pertaining to their apps. Users can easily manage all their apps by creating different repositories. Furthermore, the repositories can be edited or deleted according to the users's requirements. Another users can search for these published apps, download and install them and can manage their installed apps.

1.2 Requirement Specification

In this section, the list of features of Appstore and all the functional requirements associated with each feature has been described briefly. The important features of Appstore are the repositories management and App management. The user should be able to manage his own repositories as well as the contents of the repositories which are nothing but Apps. Also, different views of the GUI play an important role as the user of Appstore will be able to interact with the system via GUI.

1.2.1 Repositories for Appstore

Description

Appstore for LibreSocial has to developed in such a way that every user of Appstore can create and manage more than one repositories. Repositories are nothing but folders in which different Apps can be uploaded/published along with the information pertaining to Apps. This feature has the highest priority when developing an Appstore.

Functional Requirements

Following are the list of requirements associated with this feature.

1. A user of Appstore can create, rename and delete repositories.

2. A user can share his repositories with other users.
3. A user can manage the contents of his own repositories.
4. Particular user should have his own repository-list.
5. Particular repository should have all the list of users who can access it.

1.2.2 Apps in a Repository

Description

The second important feature of Appstore is Apps management. All the apps published in the Appstore are stored in a repository. So, Apps are basically the repository contents. Each App constitutes of App information(meta-data) and a .jar file(for App installation). Each App meta-data contains name, short description, category, version, keywords and an App image.

Functional Requirements

Following are the list of requirements associated with this feature.

1. A user can publish/create, edit and delete App meta-data.
2. A user can share the Apps with repositories.
3. A user can upload/download or install Apps.
4. A user can delete the installed App.

1.2.3 Publisher View

Description

When a user of Appstore has a role of a publisher, all the interaction of the user with the system is done via Publisher View of Appstore. The Publisher View is a part of graphical user interface and this feature allows the user to create and manage his apps and repositories.

Functional Requirements

Following are the list of requirements associated with this feature.

1. It can allow the user/publisher to create a new repository.
2. It can allow a user to edit/rename an existing repository.
3. It can allow a user to delete an existing repository and all its contents(Apps).

4. It can allow the user to share the repositories with others in the network.

5. User/publisher can publish a new App through Publisher View.

6. User/publisher can edit or delete an existing App through Publisher View.

7. Publisher View will show the list of all the repositories of logged in user/publisher.

8. Publisher View will show the list of Apps in a particular publisher repository.

1.2.4 Shared Repositories View

Description

When a user wants to join a shared repository of other users in the network, he uses this view. The 'Shared Repositories View' allows the user to view the list of shared repositories. The user gets the ID of a particular repository which he wants to access and pastes it in the text-area of this view. Thus, he is then able to access other user's repositories and its contents.

Functional Requirements

1. It can allow the user to join a shared repository.

2. It will show the user list of all repositories and all its contents(Apps) shared with him.

3. It can allow a user to delete a shared repository and all its contents(Apps).

4. It can allow the user to download/install Apps from the shared repositories.

1.2.5 Installed Apps View

Description

This View is required to view the list of all the Apps which the user has installed. In this view the user can manage his own installed apps.

Functional Requirements

1. This view will show the list of all the Apps installed by the user.

2. User can delete already installed Apps through 'Installed Apps' View.

1.3 Role definition for users of appstore

1.3.1 Role of a Publisher

When the user of appstore is a publisher, he mainly is responsible for creating a repository accessible only by himself. Each publisher has a right to manage his own repository. A publisher can create more than one repository and can customize them. Once the repository has been created, the publisher can rename or delete the repository. The publisher can publish apps in the repositories. The apps in the repositories contain meta-data along with the executable .jar file. The .jar file can be uploaded from the publisher's local system along with the meta-data. The app meta-data mainly contains the name, description, keywords, app size, version etc., which can later be updated if required. Only the publisher can delete his published apps from the network. Also, all the repositories of a publisher are public in nature i.e. other users can access the repository contents if they have the repository ID. Thus sharing the repository is possible if the ID of a particular repository is made public(shared via chat, email, message or wall post).

1.3.2 Role of a User

The user of appstore can view the list of Apps installed by him in the 'Installed Apps View' of appstore. User can select the app he wishes to install from repositories(either shared or his own). The user has rights to delete the apps which he has installed.

1.4 Organization

The complete thesis documentation is segregated into 7 main chapters. Chapter 1 contains a brief introduction, motivation, requirement specification and a typical use-case scenario for Appstore. Chapter 2 highlights the constructs of LibreSocial, its architecture, peculiar features in a nutshell. Also, it provides a short description about OSGi Framework. Chapter 3 gives overall feature description about different functionalities implemented in Appstore. Chapter 4 talks about the design and implementation of Appstore. Chapter 5 covers the testing of the application along with the test results. Chapter 6 talks about future work and enhancements possible for Appstore. Chapter 7 concludes the master thesis.

2 Background

These days, majority of online social networks are based on client-server technology. Thus, huge servers are required to be maintained in order to handle the user data. More the number of users, more is their data to be managed on servers. However, maintenance of these servers requires high operational costs despite of their performance. Alternatively, the applications based on P2P technology such as Skypeskype shifts the maintenance and operational costs to the users and yet is so popular for online communication. Moreover, Skype is an example of completely decentralized P2P application that provides effective file sharing and audio-video streaming at low cost. Based on P2P architecture, it would be great to have an online social network which can provide all the features similar to current online social networks such as Facebook, Twitter, LinkedIn etc. LibreSocial earlier known as LifeSocial.KOMGraffi u. a. (2010) is one such application based on P2P technology aiming to provide a distributed platform for online social networking. In this chapter, overview of LibreSocial and its features have been discussed in a nutshell. Moreover, LibreSocial has plugin-based layered P2P architecture consisting of many independent plugins. The plugins in LibreSocial are OSGi bundles. Since appstore for LibreSocial will also consist apps which are nothing but plugins, an overview of OSGi bundles and OSGi framework is also explained in one of the subsections.

2.1 What is LibreSocial?

As mentioned earlier, LibreSocial provides functionality similar to online social networks but completes differs in its architecture and online communication approaches. LibreSocial is totally distributed in nature and is completely extendible by adding additional plugins. Since LibreSocial is based on P2P technology, the users can have the benefits of secure communication and distributed storage. Functionalities like user profiles, friend lists, online messaging, writing emails, uploading and sharing photos, audio files, videos and other files, user groups and forums, notifications, voting, status updates, wall entries etc., are implemented in LibreSocial in the form of different plugins. LibreSocial has a layered architecture consisting of many independent plugins. the application is extendible by adding more plugins to it. The architecture is based on OSGi framework and the plugins are nothing but OSGi bundles that can be loaded and unloaded during run time. Thus, it becomes quite manageable to add new plugins or update the existing plugins of the application.

As said earlier, LibreSocial has a layered architecture with the lowermost layer consisting of structured P2P overlay known as FreePastryFreePastry which provides ID-based routing among the peers. FreePastry allows a reliable storage of data in the network. The layer above the P2P overlay is the framework layer in which PASTRowstron und Druschel (2001) is used for secure replication. The topmost layer consists of the GUI(Graphical User Interface)/Plugins. Since LibreSocial is a web-based application, it uses the standard web technologies like HTML, CSS, Ajax, Boostrapbootstrap, Knockback.jsknockback and JQueryjquery. The architecture of LibreSocial comprising of all the above mentioned components is given in the figure 1.

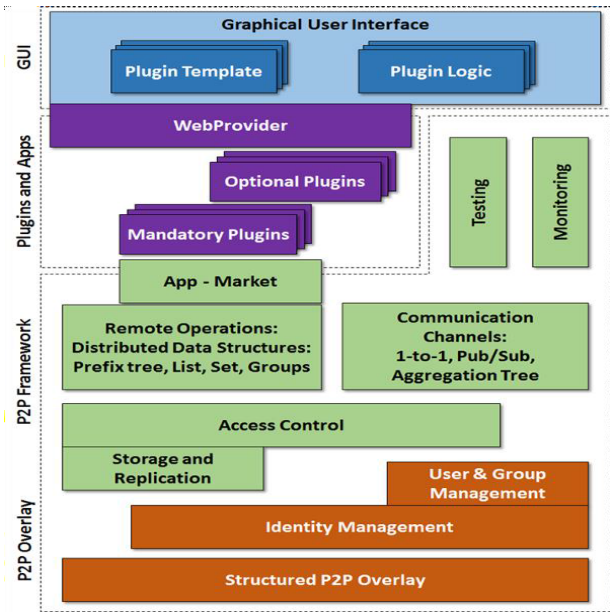


Figure 2.1: New modified architecture of LifeSocial (PG-2013/14)

.

2.1.1 LibreSocial- Features in a nutshell

In this section, different parts of LibreSocial are explained in brief. There are three main layers namely 1. GUI and plugins, 2. Framework and 3. P2P overlay. Moreover, to observe the quality of service of the application, LibreSocial uses a java based standalone application for monitoring called SkyEye.KOM. All these features are described below.

2.1.2 GUI/Plugins

LibreSocial used latest web technologies like HTML 5, CSS, bootstrap, JQuery and Knockback.js accessible through a web browser with multi-language support. Also, a

single page design is implemented so that the chunks or HTML fragments and JSON data are loaded instead of full HTML page during every request. GUI/Plugin layer is the topmost layer in LibreSocial which is interactive and gives a commercial look to the application similar to other online social networks. Components of GUI are as seen in the figure 2. Main parts of GUI layer are 1. Plugin Part 2.Webprovider plugin 3. REST Handler and 4. HTML architecture. .

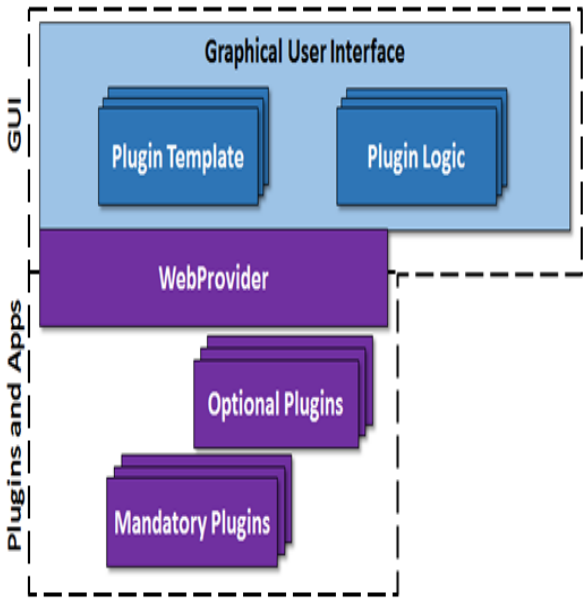


Figure 2.2: GUI layer with plugins (PG-2013/14)

Plugin Part Each plugin consists of a plugin template and a plugin logic. The plugin template is nothing but a simple HTML file template which uses Knockback.js. Knockback is a standalone implementation with Model-View-View-Model(MVVM)modelparadigm. The benefits of using Knockback.js is that it s small, lightweight and efficiently used for binding each element to a model. This implementation helps in tracking the dependency if any value of the element changes. Plugin logic is mainly about taking the user events from front-end and sending it to REST Handler via Webprovider plugin and the data is again returned from REST. Thus with the help of plugin logic, the data i rendered to desired plugin template. This works by creating Models, Collections and Views(according to MVVM paradigm). All these are created in .js files for each plugin. For Models and Collections backbone.js backbone is used. Knockback.js is used for Views. A Model is a basic set of functionality which contains mostly the interactive parameters and Collection is an ordered set of Models. whenever there is a change in the attribute of a model, it can be seen in the View part. Detailed information can be seen in PG2013 (2013)

Webprovider plugin Webprovider is a plugin through which the APIs of other plugins can be accessed via REST interface(more information about REST is given in the next subsection). It consists of an embedded server and so all the dynamic requests are handles inside the browser only. The Webprovider plugins acts as an interface between other plugins and the static files such as HTML, CSS and JavaScript files requested by the browser as shown in figure 3. All the plugin calls are made through Webprovider.

.

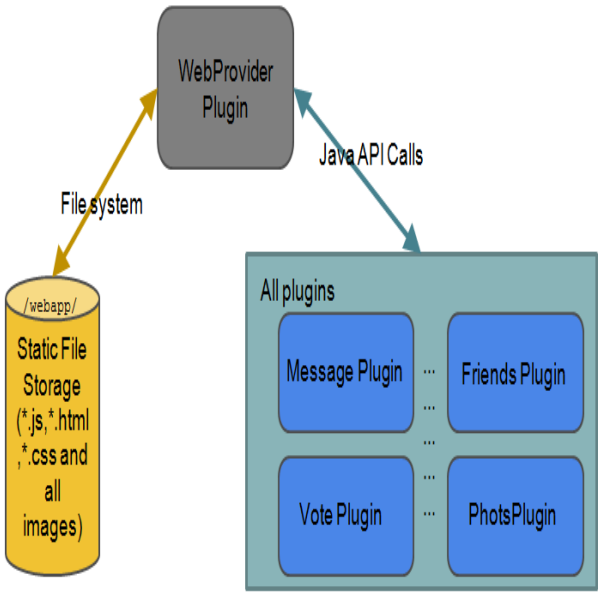


Figure 2.3: Webprovider plugin (PG-2013/14)

The structure of the Webprovider plugin and its interaction with other components are as shown in the figure 4. The Webprovider contains both java classes as well as the static files such as HTML files and JavaScript files. As shown in the following figure, Webprovider plugin uses the data from each plugin via REST Handler. Every plugin has an associated PluginHandler which handles the requests from the browser. To be precise, the requests are from the browser are handled via REST by 'PluginHandler' class. The plugins can be accessed from PluginHandler class with 'getPluginProvider' method. For example, as given in the figure, the Login plugin can be accessed by 'LoginPluginHandler' class with 'getPluginProvider' method. In a similar manner, each plugin has its own handler class associated with it which receives and understands the request from the browser. If there is data expected as a response to the request, the data from the plugin is serialized to JSON in the PluginHandler and transmitted to the browser. For events that are sent from the back-end application to the browser web app, web-sockets are used as a unidirectional channel. For static files of the web application the WebAppFilesHandler class looks up the resources folder of the web provider project and delivers the files to the browserPG2013 (2013).

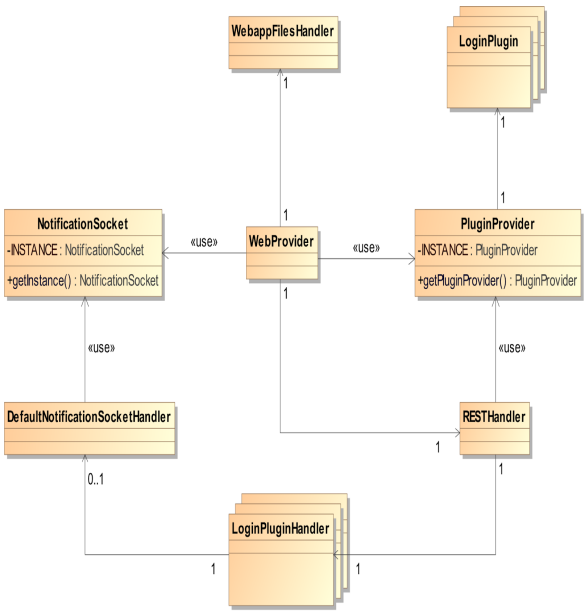


Figure 2.4: Structure of Webprovider Plugin (PG-2013/14)

.

REST Handler

The REST Handler basically handles the requests from the browser. For example, getLoginID, createUser, deleteFile, uploadPhoto or getting any kind of data from the network are the requests from the front-end. RESTHandler is a part of Webprovider plugin which is used to access the API of other plugins. REST Handler may consist of several if/else blocks or switch statements and the requests are made in the for of urls like baseRequest/target i.e group/createNewGroup. Here the baseRequest is group and the target is to create a new group. The result of the this call is transferred to the browser using response.

HTML Architecture

The GUI for LibreSocial has been implemented using HTML5 and JavaScript technologies. The GUI is completely decoupled from the actual user interface of the application and hence allowing it to be accessible on mobile devices as well as different P2P systems. The application uses a single page approach as mentioned in earlier sections as it has its own advantages. The browser to back-end communication can be seen as in the figure given 5. During the first connection, HTML and CSS contents are sent from the application to the browser. Since all the elements are static, this happens in the beginning when the user opens the GUI. The other connections mainly deal with the requests and response from the browser to plugins in the back-end respectively via REST interface. In this manner, the

GUI of each plugin is able to interact with the plugin back-end. The response is processed and transferred into JSON serialized data usually as in JavaScript for efficient parsing. The communications that are initiated by the back-end like status updates, incoming messages, email notifications or other notifications are handles by the websockets. Websockets provide a simple way to send data from the HTTP server implementation to the browser. .

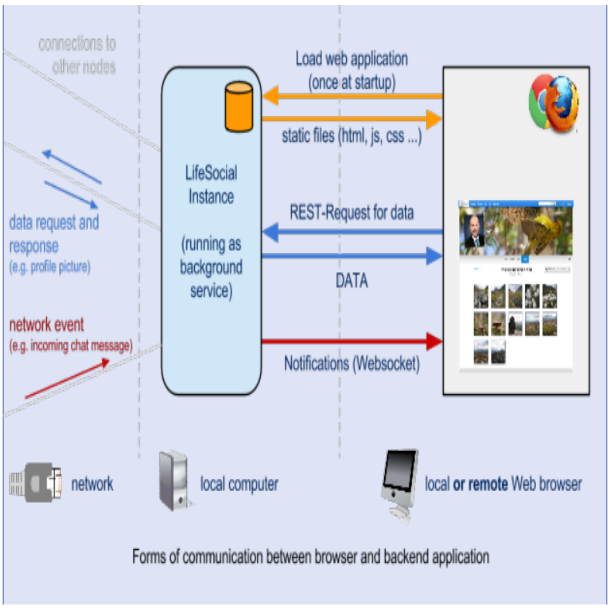


Figure 2.5: Browser-Backend communication(PG-2013/14)

2.1.3 Framework

The framework layer is responsible for reliable storage and secure replication of data items in the network. Important data structures, like distributed lists, access control secure as well as different communication options are implemented in this layer. The replication manager, provided by PAST runs replication maintenance operation on defined time intervals to ensure that for any data item with key K, R replicas are maintained. The uppermost layer of GUI and plugins communicate with the framework through a reliable interface. Secure replication ensures that the data is still available even if the peer node responsible for the requested data goes offline. Currently, PAST uses a (k,r) replication scheme. In such a scheme for a given data object with a key k, PAST ensures r replicas are made available of the data item with key k. PAST replicates the data item to the leafset nodes of the responsible peer and thereby maintains r replicas for a data item of key k in the networkPG2013 (2013).

Data structures are important to handle data in an appropriate way since they offer specific solutions how to organize and access data efficiently. Presently, a distributed

list, a distributed set, a prefix hash tree and a distributed tree are implemented in the application. Since all these data structures are distributed in nature, they provide interfaces to manage and access the data in a distributed networkPG2013 (2013).Since the data structures are distributed in nature, it manages the data and distributes the data over the nodes in the network. The access controls are enforced by encrypting the buckets with cryptographic keys. The main user IDs are stored in the form of cryptographic public keys to uniquely identify the users and allow their authentication to access their objects in the network. Thus, the information stored in the network is completely encrypted and can only be readable or modified if one has the public key from its owner. The distributed tree uses the distributed link lists. In this data structure, each node is stored in a distributed linked list which is split up into several buckets. Each buckets refers to the distributed linked list where its child nodes are storedPG2013 (2013). Different components of framework layer are as shown in figure 6: .

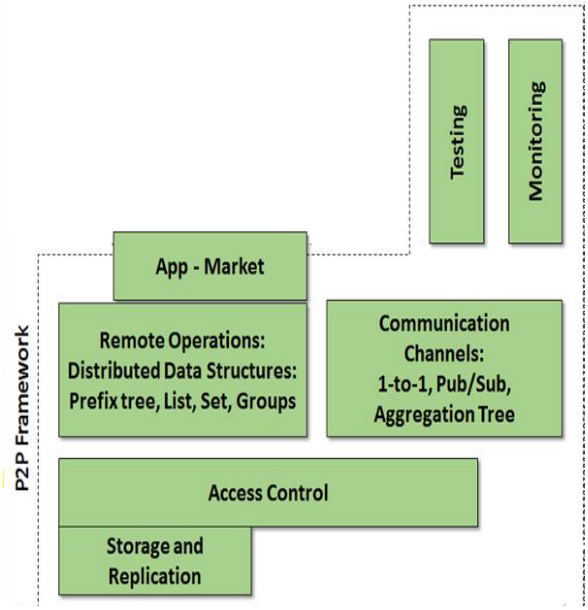


Figure 2.6: Framework layer(PG-2013/14)

2.1.4 P2P Overlay

LibreSocial uses a structured P2P overlay called FreePastry for IT-based routing among the participating nodes. The parts of overlay layer can be seen in the figure 7. FreePastry is an open-source implementation of a structured P2P overlay. Data objects and participating nodes are identified by global unique ID and all the objects are stored by the nodes with their respective IDs. A message overhead of O(log)N is produced while storing and retrieving data in the networkGraffi u. a. (2010). To provide secure communication, asymmetric cryptography is used for encryption and

decryption of data. The public keys are generated based on cryptography server as node IDs. There is a mapping-item which is used to map node ID to the user ID in order to give a unique identification to the user. The routing mechanism by P2P overlay is used to retrieve this mapping-item. The mapping-item stored in the network helps in searching the user based on his user ID if other peers want to communicate to him. P2P overlay also helps in parallel and iterative routing in order to avoid impact of malicious nodes. Moreover, a node responsible for sending the data objects keeps the track of requesting nodes. It this way, the node responsible to send data acts as a cache and stores different metrics like memory info, bandwidth etc.
.

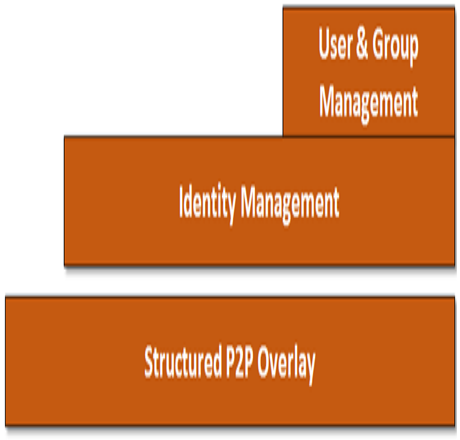


Figure 2.7: P2P Overlay layer(PG-2013/14)

2.2 Overview of OSGi Framework

The OSGi (Open Service Gateway initiative) is a specification describing a modular system and a service platform in Java. Unlike standalone Java/VM environments, a dynamic component model is implemented and the applications (called bundles) are remotely installed, started, stopped, updated and uninstalled. LibreSocial implements this dynamic component model and supports the modular applications as bundles. Bundles are regular jar components having manifest headers. Each bundle is tightly coupled collection of Java classes, jars, and config files, etc. and can be loaded dynamically. The plugin-based architecture of LibreSocial is built around the bundles in OSGi. The LibreSocial framework contains many bundles and also supports an extensible architecture to accommodate different plugins as bundles such as games, collaborative applications, etc. It is able to deploy these bundles in the network without needing a central repository. The idea of introducing OSGi bundles is to get familiar with the appstore functionality of providing apps which

are OSGi bundles loaded and unloaded during runtime. The next subsection talks about the basic OSGi structure and OSGi bundles as plugins to LibreSocial.

2.2.1 OSGi structure

The basic OSGi architecture is as shown in the figure 8. The OSGi components rest on the JVM(Java Virtual Machine) and provide mechanisms for service management, component definition, execution, management and life cycle control of modules. The most important concepts of OSGi are as follows: .

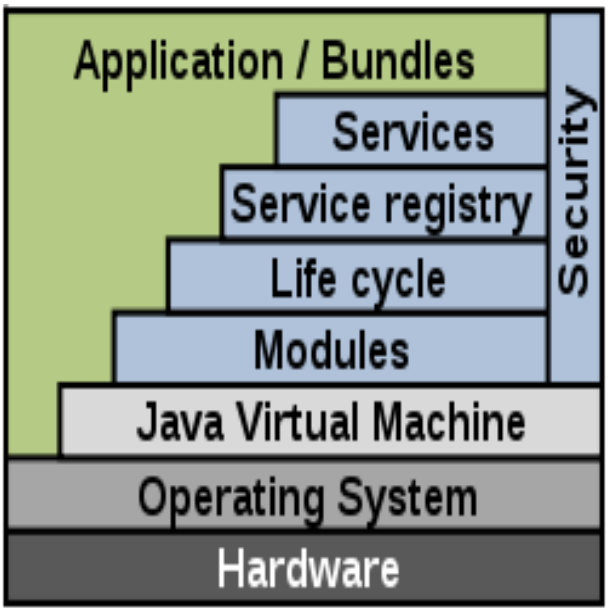


Figure 2.8: OSGi Architecture(Wikipedia.org)

1 Bundle:
In OSGi, a bundle can be described as a components that builds the structure. Basically each OSGi bundle is a jar file. But unlike jar files the OSGi bundles have OSGi specific manifest definition and OSGi specific classes. As mentioned above, a bundle is a jar file consisting of at least an Activator class and a MANIFEST file with OSGi specific information. Some of the important information according to osgi about bundles may be:

- Bundle-Name: The name of the bundle.
- Bundle-Symbolic Name: As the only mandatory definition in the MANIFEST file, symbolic name defines the unique name of the bundle in OSGi ecosystem. As this definition should be unique, it is generally defined as the base package name of the bundle by convention.
- Bundle-Description: The description of the bundle.

- Bundle-Manifest Version: The manifest version of the bundle.
- Bundle-Version: OSGi bundle version.
- Bundle-Activator: This class is used to control the bundle's life cycle. This class is called by the OSGi to start or stop the bundle.
- Export-Package: Packages that is wanted to be used by other bundles are defined in this section.
- Import-Package: Packages that is needed for the execution of current bundle are defined in this section.

2 Services:
Services can be described as interfaces which provide interaction between different bundles of the structure. services are registered with an implementation that performs an interface. Thus the access through OSGi services makes the OSGi based systems more loosely coupled as compared to normal jar based java structures. These structures also make it possible to change the components during run-time. OSGi implements a service directory for services to be register and accessed through and also provides mechanism to manage those services.

3 Life Cycle:
OSGi also provides a platform to control the life-cycle of bundles. Each OSGi bundle has its own OSGi configuration in terms of dependencies and system run by OSGi itself. All OSGi bundles compose of a configuration file with an order and life-cycle management for each of the components is applied in that very order. This is controlled by the ""'Activator Class"" ' which has to exist in every OSGi bundle. As given in osgi, the life-cycle steps of a bundle are

- INSTALLED: This state indicates that the installation step has been successfully completed. In this case, neither dependency analysis nor the class loading is made. Only required steps are performed, such as defining bundle properties analysing its Manifest file.
- RESOLVED: Bundle is found in this state when OSGi resolves and satisfies all of its dependencies and makes class loading operations. This is the state that comes before starting and after stopping.
- STARTING: This is the state that bundle is found when the "start" method of the Activator of the bundle is called, but not yet as successfully or unsuccessfully finished.
- ACTIVE: The bundle is successfully started and running meaning the "start" method of the Activator resulted success.
- STOPPING: This is the state that bundle is found when the "stop" method of the Activator of the bundle is called, but not yet as successfully or unsuccessfully finished.

- UNINSTALLED: This is the state when the bundle is removed from the system. In this situation, there is no transition to another state. The component must be installed again.

The complete life-cycle and the above mentioned steps are shown in the figure 9: .

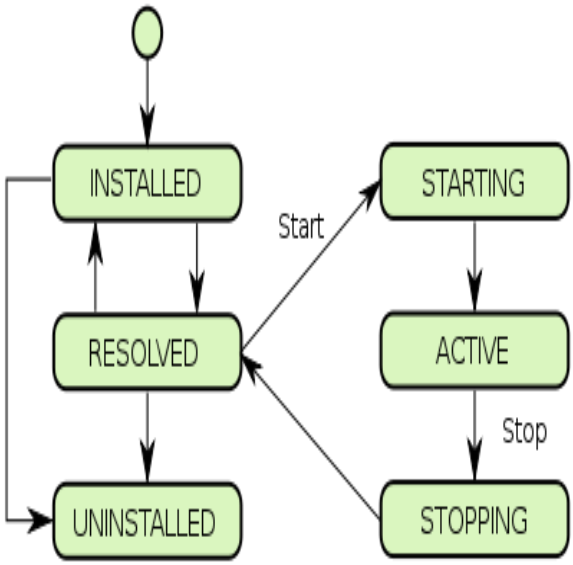


Figure 2.9: Bundle life cycle(Wikipedia.org)

2.2.2 OSGi bundles as plugins to LibreSocial

LibreSocial is an application which is based on OSGi platform. It is therefore, very obvious that LibreSocial contains all the plugins in the form of OSGi bundles. The architecture of LibreSocial allows us to add or remove bundles at run-time. Such a type of architecture is suitable from social networking point of view. LibreSocial currently contains essential features of an online social network such as profile, friend lists, file transfer, groups and forums, photo albums, voting etc., which are technically nothing but OSGi bundles. However, many other features like games, collaborative work, app-market etc., can be provided as independent further extensions to the existing framework of LibreSocial.

.
All the plugins used via OSGi contain a definition file consisting of provided services and important references. This helps the system to resolve the dependencies between the plugins. In order to use the services of the plugins, an interface class has to be created which in the case of LibreSocial is the Webprovider plugin. Thus, Webprovider contains plugin handlers registered with REST Handler class for each plugin. More information about how to create a plugin in LibreSocial can be found

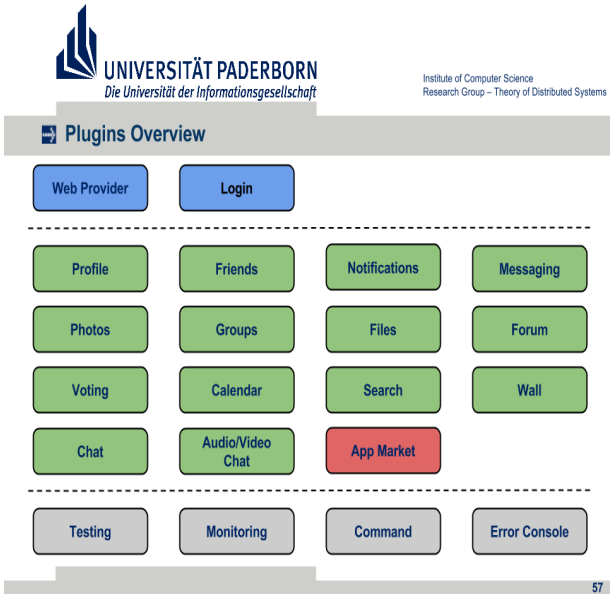


Figure 2.10: Final presentation(PG-2013/14)

in the documentation of previous PGs i.e. PG2013 (2013) and PG2012 (2012). Figure 10 lists all the important plugins of LibreSocial.

Appendix

...

List of Figures

2.1 New modified architecture of LifeSocial (PG-2013/14) 8
2.2 GUI layer with plugins (PG-2013/14) 9
2.3 Webprovider plugin (PG-2013/14) 10
2.4 Structure of Webprovider Plugin (PG-2013/14) 11
2.5 Browser-Backend communication(PG-2013/14) 12
2.6 Framework layer(PG-2013/14) 13
2.7 P2P Overlay layer(PG-2013/14) 14
2.8 OSGi Architecture(Wikipedia.org) 15
2.9 Bundle life cycle(Wikipedia.org) 17
2.10 Final presentation(PG-2013/14) 18

Bibliography

[backbone] *Backbone.js.* – http://backbonejs.org/

[bootstrap] *Bootstrap.* – http://www.getbootstrap.com/

[FreePastry] *FreePastry.* – http://www.freepastry.org/FreePastry

[jquery] *Jquery.* – http://jquery.com/

[knockback] *Knockback.js.* – http://kmalakoff.github.io/knockback/

[model] : *Model View ViewModel(MVVM).* http://en.wikipedia.org/wiki/Model_View_ViewModel/

[osgi] : *OSGi: An Introduction.* http://www.javacodegeeks.com/2012/01/osgi-introduction.html/

[skype] : *Skype.* http://www.skype.com/en/

[PG2012 2012] : *Final Report: Project Group 2012. A Peer-to-Peer Framework for Social Networks.* 2012

[PG2013 2013] : *Final Document: Project Group 2013. A Peer-to-Peer Framework for Social Networks.* 2013

[Graffi u. a. 2010] Graffi, Kalman ; Gross, Christian ; Mukherjee, Patrick ; Kovacevic, Aleksandra ; Steinmetz, Ralf: LifeSocial. KOM: A P2P-based platform for secure online social networks. In: *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on* IEEE (Veranst.), 2010, S. 1–2

[Rowstron und Druschel 2001] Rowstron, Antony ; Druschel, Peter: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: *Middleware 2001* Springer (Veranst.), 2001, S. 329–350