

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Fakultät für Elektrotechnik, Informatik und Mathematik

Master thesis

Distributed Access Control Mechanisms for Peer-to-Peer Networks

Christy Jovita Bibiana Arockiaraj
Manasa Veena Rapolu
Student ID:
6750106
6727347
E-Mail:
achristy@uni-paderborn.de
mvrapolu@uni-paderborn.de

Paderborn, February 26, 2015

supervised by
Jun.-Prof. Dr.-Ing. Kalman Graffi
Prof. Dr. Christian Scheideler

Declaration

I hereby declare that this thesis incorporates the result of a joint research undertaken in collaboration with Christy Jovita Bibiana Arockiaraj and Manasa Veena Rapolu under the supervision of Junior Professor Kalman Graffi. The collaboration and individual work is covered in Chapter 1 of this thesis. In table of contents the chapters, sections and subsections that are colored in blue are written by author Manasa Veena Rapolu and similarly dark gray colored contents are written by Christy Jovita Bibiana Arockiaraj. We have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Paderborn, February 26, 2015

Christy Jovita Bibiana Arockiaraj
Manasa Veena Rapolu

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 2
  - 1.2.1 Hierarchical Structure . . . 3
  - 1.2.2 Functionalities . . . 4
  - 1.2.3 Access Control Mechanisms . . . 4
  - 1.2.4 Testing . . . 4
- 1.3 Thesis Outline . . . 5
- 1.4 Collaborative Work . . . 6

2 Background **7**
- 2.1 Peer to Peer technology . . . 7
- 2.2 OSGi Framework . . . 7
- 2.3 Lifesocial . . . 8
  - 2.3.1 Overlay . . . 10
  - 2.3.2 Framework . . . 10
  - 2.3.3 Plugins . . . 11
  - 2.3.4 GUI . . . 11

3 Design Of Group-based Plugin **13**
- 3.1 Groups Plugin . . . 13
- 3.2 Create Community . . . 16
- 3.3 Create Inner-level Communities . . . 17
- 3.4 Add Folder . . . 18
- 3.5 Leave Community . . . 19
- 3.6 Access Control Mechanism . . . 19
- 3.7 Adding Members To Community . . . 20
- 3.8 Deleting An Item . . . 21
- 3.9 Renaming Community . . . 21
- 3.10 Uploading/downloading Files . . . 22

The chapters, sections and subsections coloured in blue are written by Manasa Veena Rapolu and dark gray are written by Christy Jovita Bibiana Arockiaraj

3.11 Get inner level communities . . . 22
3.12 Get Contents Of A Community . . . 23

4 Design Of Graphical User Interface **25**
4.1 Design of Top-level Communities . . . 26
4.2 Design of Inner-level Communities . . . 26
4.3 Navigation Through Breadcrumbs . . . 28
4.4 Navigation To Inner Contents of Hierarchy . . . 28
4.5 Adding Members . . . 29
4.6 Adding Friends . . . 29
4.7 Display Members . . . 30
4.8 Upload/Download Files . . . 30
4.9 Delete . . . 31
4.10 Rename . . . 31
4.11 Leave Community . . . 32

5 Final Testing **35**
5.1 Test Plan . . . 35

6 Conclusion **51**

Appendix **53**

Bibliography **55**

1 Introduction

1.1 Motivation

Social networks became vital part of one's life. Famous social networking sites like Facebook, MySpace, Twitter bring people from all over the world together onto a common platform allowing them to get to know about each other. This could include families, friendship and group relations. Sharing any kind of emotions like happiness, love or sorrow are easily expressed with people no matter how far they are from each other. Thus, they stand as quick messengers. Many features like one's personal profile, collection of photos and videos, user groups or communities, blogs, and many more are collectively included in one social networking website. People not only use these social networks to improve their personal relationships but also to enhance their professional life and develop business contacts. Websites like LinkedIn, XING serve as a convenient place for professionals to develop their contacts and talk about business. Moreover, they serve as a great platform to find employment. All these social networks are centralized i.e they are based on a central server. Centralized social networks suffer from certain issues like concerning over security, high maintenance and highly expensive costs. They are also prone to single point of failure in which complete system breakdowns due to failure of the central server.

Peer-to-Peer (P2P) technology stands out as a solution to the problems caused by centralized systems, as there is no central server in a P2P network. It is completely decentralized in which each peer in the network acts as both client and server, sharing their CPU power, bandwidth and storage capacity . The data is shared among all the peers and distributed over the network instead of storing at a central repository. This results in saving huge amount of money for maintaining servers and there is no problem of single point of failure that can occur in P2P networks. Moreover, P2P technology do not suffer from the problem concerning the security of data of the users as all their information is kept secure and private within the network.

LifeSocial is one such social network based on P2P technology which is devel-

oped by Technische Universitat Darmstadt, followed by projects at University of Paderborn. LifeSocial application corresponds to any other social networks like facebook in which most of the features like profiles, photos, video/audio chat, messages, chat, groups, apps are developed. As it is a developing application, several functionalities needs to be developed and extended further. In this thesis, we extend the groups plugin to be used in E-learning and Social Intranet use cases.

1.2 Goals

The main aim of this master thesis is to extend the Group based plugin. Previously, using groups plugin one can create groups only on single level i.e groups inside groups was not possible. Users can add several folders or files to the group but cannot add sub-group inside a group. This means that members can be added to the entire group and can view all the folders and files in that group. Also, access control mechanisms were not implemented at each level. They were implemented only to one level i.e only the owner of the item is able to modify its content and the others do not. And adding members to a group can be done only by becoming friends with those members and then add them to the group.

In this thesis, the main idea is to allow the users to create groups inside groups in which the user can create a sub-group or sub-groups inside the main group than just creating folders or files inside a group. This enhancement enables creation of sub-groups inside groups which makes adding members to either to main group or to sub-groups possible. People who are added to sub-groups can only view the contents of that sub-group without able to see the main group and its content. Whereas, people who are added to inner sub-groups can view only that particular sub-groups and cannot view the top-level groups. This enables restricting certain members to be added to certain sub-groups. Other thing is to implement the access control mechanisms i.e members can be added to each group with either read or write access. The access control mechanisms have been implemented in this thesis using which one can restrict the access of a member who is being added to the group. Also, implementing the functionality of adding non-friends to any group is aimed at. An example use case scenario involving three users and the access control mechanisms for each user are shown in the Figures 1.1 and 1.2.

In this master thesis, careful consideration of certain goals are analyzed that are discussed below.

ACL Mechanisms :

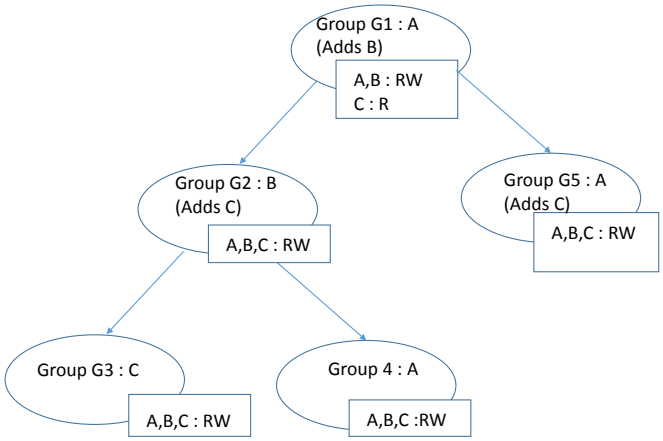


Figure 1.1: Access Controls

USECASE SCENARIO

Steps	User A Page	User B page	User C page
1	Create group G1(RW)		
2	Add user B	Group G1(RW) is added	
3		Creates subgroup G2(RW) under G1(RW)	
4	Group G2(RW) is added under Group G1(RW)		
5		Adds user C to subgroup G2(RW)	Group G1(R),G2(RW) is added
6			Creates subgroup G3(RW) is created under G2(RW)
7	Group G1(RW),G2(RW),G3(RW) is added	Group G1(RW),G2(RW),G3(RW) is added	Group G1(R),G2(RW),G3(RW) is added
	User can rename and delete G1	User can rename and delete G2	User can rename and delete G3

Figure 1.2: Use Case Scenario

1.2.1 Hierarchical Structure

First goal is to design the hierarchical structure of how exactly the whole use case environment looks like. Every group can consist of a group, folder and a file i.e inside a group, an other new group, folder or a file can be created.

Members can be added to groups at different level. An example scenario is explained as; when user A creates a main group G1 and adds user B to it, and user B creates a sub-group G2 in it and adds user C to the sub-group and user C creates G3 inside G2 then, user A should be granted read/write access to all the sub-groups created inside the main group and user C should be granted only read access until the main group. This means that user C should be able to view all the parent groups until the main group i.e G2, G1 and user A should view all its child groups i.e G2, G3 with both read and write access as shown in figures above.

1.2.2 Functionalities

Second goal is to analyze the functionalities that needs be written at each level in this hierarchical structure. Different functionalities like creating a new community inside a top-level community, adding a folder to the main and inner communities, uploading and downloading a file, adding members to the communities, renaming communities and folders and files, leaving communities, deleting folders and files. These are the primary functionalities which needs to be implemented in this thesis.

1.2.3 Access Control Mechanisms

Third goal is to set the access controls at each level. Members should be added either with read or write access to any community. Members who are added with read access can only view the contents of that community and cannot see its inner communitites. Whereas, members who are granted write access are allowed to view and modify the contents of that community and can see its inner communities.
For example in the university scenario, read/write access rights can be granted by the owner to members of the community depending on their designations. For instance, an assistant who works under the professor is granted write access to the community created by the professor where he can add materials and tutorials and manage the content. Whereas, students are granted read access to communities where lecture materials are uploaded in which they can just view the content and write access is granted to communities where they have rights to submit their home work assignments. This way access controls are set for the members of different communitites.

1.2.4 Testing

Final goal is to test the environment with professors and students at the university. This enhances the application to be used at universities in real time. Any errors that occur during testing should be rectified and corrected accordingly.

1.3 Thesis Outline

The structure of the thesis is presented in this section along with the contributions of the two authors.

Chapter 1

This chapter presents the motivation, goals and the outline of the thesis. This chapter is written by the author Manasa Veena Rapolu.

Chapter 2

This chapter presents a theoretical knowledge on Peer-to-Peer technology, OSGi Framework and `LifeSocial` architecture in order to provide brief ground knowledge to the readers. The architecture of `LifeSocial` is layered which consists of P2P overlay, P2P framework, plugins and app, and Graphical User Interface (GUI). All these layers are explained in this chapter to give an overview to the reader. This chapter is written by the author Christy Jovita Bibiana Arockiaraj.

Chapter 3

This chapter presents the implementation of different functionalities as an extension to group based plugin. The first six functionalities are written by the author Christy Jovita Bibiana Arockiaraj and the next six functionalities by the author Manasa Veena Rapolu.

Chapter 4

This chapter gives an overview about the implementation of the graphical user interface for LifeSocial. This is achieved using different technologies like knockback.js, ajax, jquery, html5 and css. Designing the GUI for top-level communities, inner-level communities, navigation through breadcrumbs and to inner contents, adding members and adding friends are carried out and documented by the author Christy Jovita Bibiana. Displaying the members, uploading/downloading of files, deleting communities, renaming communities/-folders/files and leaving community functions are designed and documented by the author Manasa Veena Rapolu.

Chapter 5

This chapter presents an overview of the test case scenario that has been developed in order to perform testing of all the functionalities that have been

developed by both the authors. Testing has been carried out by both the authors in which each author tested the functionalities that they implemented. Documenting this work is done by the author Manasa Veena Rapolu.

Chapter 6

This chapter presents the conclusion of this thesis by concluding the work that has been carried out and the future work that can be done as a continuation to this work. This chapter has been written by the author Christy Jovita Bibiana Arockiaraj.

1.4 Collaborative Work

This thesis is a combined work of both the authors Manasa Veena Rapolu and Cristy Jovita Bibiana Arockiaraj. At first, each author got an individual thesis topic in which two different use cases needs to be designed by extending the Groups Plugin. One use case is to develop an E-learning community which serves as a medium of communication among students and lecturers at universities. Another is to develop social intranet that serves as a medium of communication among various departments in a company. Since there are many common parts in both of the thesis like, the hierarchical structure, defining access controls, etc collaborative work is thought to be a good option for achieving overall intended goals avoiding redundancy of works.
During the period of this master thesis, much work has been done in the backend (implementation of functionalities in java) and frontend (designing the GUI) by both the authors in order to develop all the intended functionalities of LifeSocial. The main hierarchical structure is designed and implemented by both the authors. After that, both the authors worked on their independent tasks in developing the functionalities which are aimed at. The functionalities that are implemented by both the authors are mentioned in the chapters 3 and 4 of this thesis. Author Manasa Veena Rapolu worked on implementing the functions like creating communities inside communities, renaming communities and author Arockiaraj worked on implementing the functionalities like adding members to communities and access control mechanisms. In table of contents the chapters, sections and subsections colored in blue are written by author Manasa Veena Rapolu and similarly dark gray colored contents are written by Christy Jovita Bibiana Arockiaraj.

2 Background

This section gives a brief overview of the background knowledge that is needed to interpret this thesis. It includes detailing about Peer-to-Peer technology, OSGi framework and architecture of LifeSocial. This chapter is written by the author Christy Jovita Bibiana Arockiraj.

2.1 Peer to Peer technology

Peer-to-peer(P2P) technology is a decentralized system in which, nodes known as peers are connected together and contribute their resources. As it is a decentralized system, there is no need of central server. In P2P technology, peers are both client and server. In case of client-server architecture, there is a central server which is responsible for providing resources to several clients. Therefore, whenever there is a problem in the server, the whole network goes down. Moreover, as number of clients increase in the client/server architecture, the server performance will decrease. However, in P2P technology, the overall performance increases with increase in peers joining the network. Each peer stores the data in a distributed manner, more likely in a replicated way of storing data in other peers. In contrast to client/server architecture, the peers may join or dis-join the network whenever they want. If a peer leaves the network, the network remains undisturbed by using the other peers connected. One of the widely used P2P applications is BitTorrent which is used for downloading the files. Other P2P applications are PPLive, PeerCast, Gnutella.
In this thesis, the P2P technology is used in an another field of social networking. Social networking has gained more popularity in the last decades. Most of the social networking applications use client/server architecture. However, it has its own disadvantages like high maintenance cost, server bottleneck and data privacy. Therefore, these limitations can be overcome by using P2P approach in social networking.

2.2 OSGi Framework

In this thesis, LifeSocial architecture is developed based on plugins. These plugins are based on Open Services Gateway Initiative(OSGi). OSGi provides

a modular architecture for today's large-scale distributed systems as well as small, embedded applications and device networks[1]. The modular feature in OSGi makes easy for the developers to develop the large system by dividing it into minor modules. LifeSocial is developed using the Eclipse OSGi. OSGi technology provides certain specifications to develop component system in java. It makes sure that the components should hide their implementations to the other components. These components are also called as OSGi bundles. These bundles can be used by other bundles only if it is exported explicitly. The components interact with each other using the *services*. These services are used for connecting the component by providing or requesting the services of the component. These specifications makes the OSGi an important technology in developing dynamic component system for java.

2.3 Lifesocial

This section gives an overview about LifeSocial. LifeSocial is a social network, which is based on distributed systems. It is based on a four layered architecture which consists of P2P overlay, P2P frameworks, Plugins and GUI. These four layers are explained in the next upcoming sections. The important feature of LifeSocial is the distributed storage of data. This gives the privilege to the user to have control of their own data rather than storing it in the centralized server. As a social network, it connects users in the network and provide various functionalities like creating user profile, sending messages, live video and audio chatting, store files, create groups and make friends. Since it is based on OSGi framework, it can be easily extended to add more functionalities and features to it. The Figure 2.1 shows the four layers in the architecture and each layers has it own features.

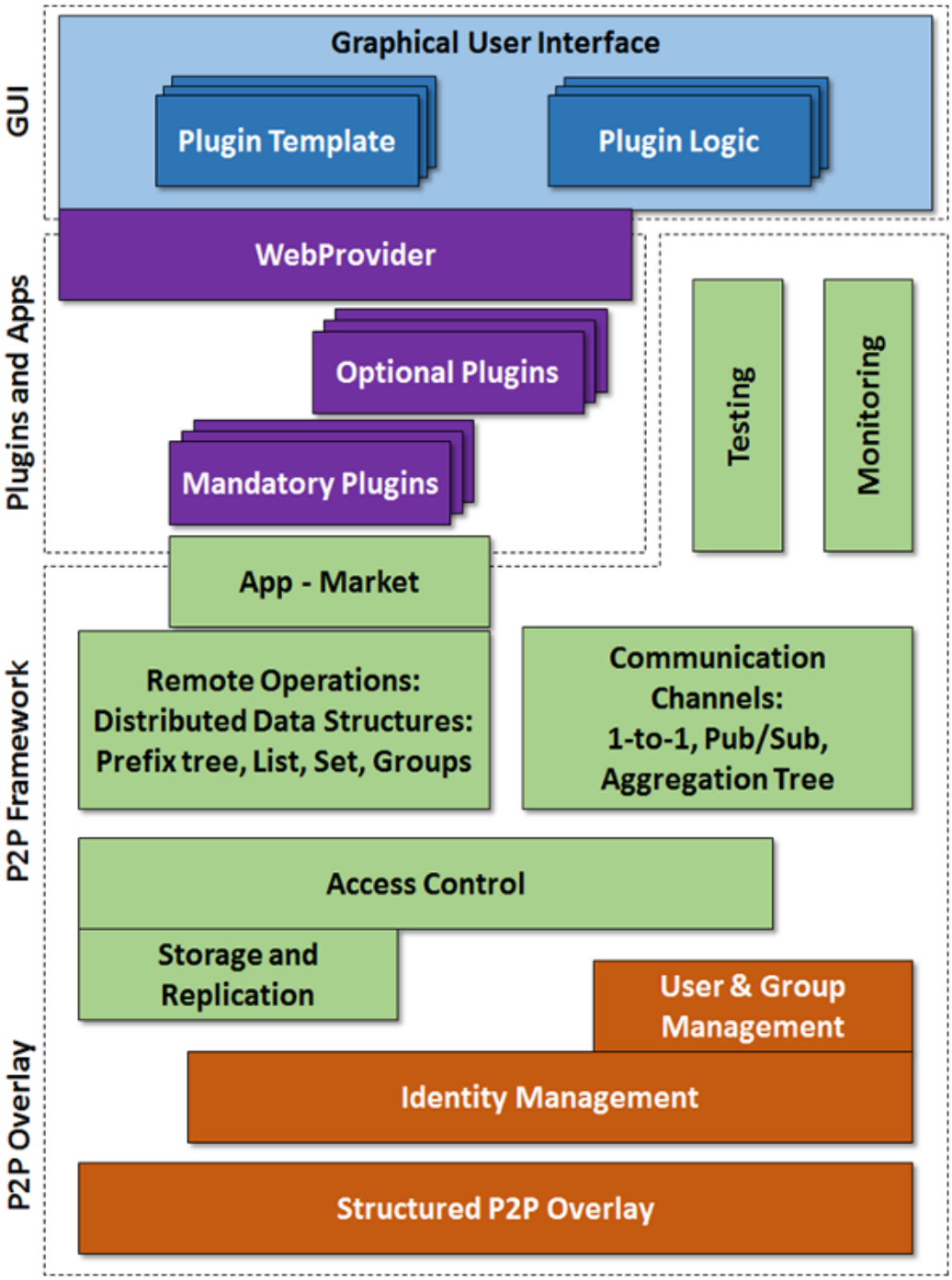


Figure 2.1: Lifesocial Architecture [2]

2.3.1 Overlay

This section describes about the P2P overlay in the architecture. The P2P overlay layer is the lower layer of the architecture, which is responsible for the peer-to-peer functionalities. The nodes in the P2P systems are decentralized and follows the Pastry mechanism for routing. These pastry nodes are organized in distributed manner. Each node and object in the pastry has its own unique identifier, which helps in routing messages to specific nodes. Pastry incorporates Id-based routing mechanism. FreePastry is an open source implementation based on Pastry, which is used in the P2P overlay of Lifesocial.

2.3.2 Framework

The layer above the P2P overlay is the P2P framework. This layer is responsible for the storage and replication as well as the access control mechanisms. It is responsible for providing the data structures like Distributed Linked list and set, which helps to organize the data being stored in the network. The functionalities in this layer is used by the immediate Plugin layer.
As the nodes are connected in the network, the storage of data among the peers is important. As the peer goes offline, it is necessary to replicate the data in the network. PAST is responsible for replicating the data among the availbale peers. It is important to take care of the retrieval and storage of data. Therefore, the replication of data among the peers should be free from the security attacks.
Next important functionality of the framework layer is the data structures. It is important to structure the data and store it in the newtwork. In LifeSocial, **Distributed Linked list** and **Distributed set** are used. These data structures divides the data into list of items with a specific bucket size and later this local list of data items are used for the operation. It is important to specify the right bucket size for splitting the data items. This is shown in Figure 2.2.

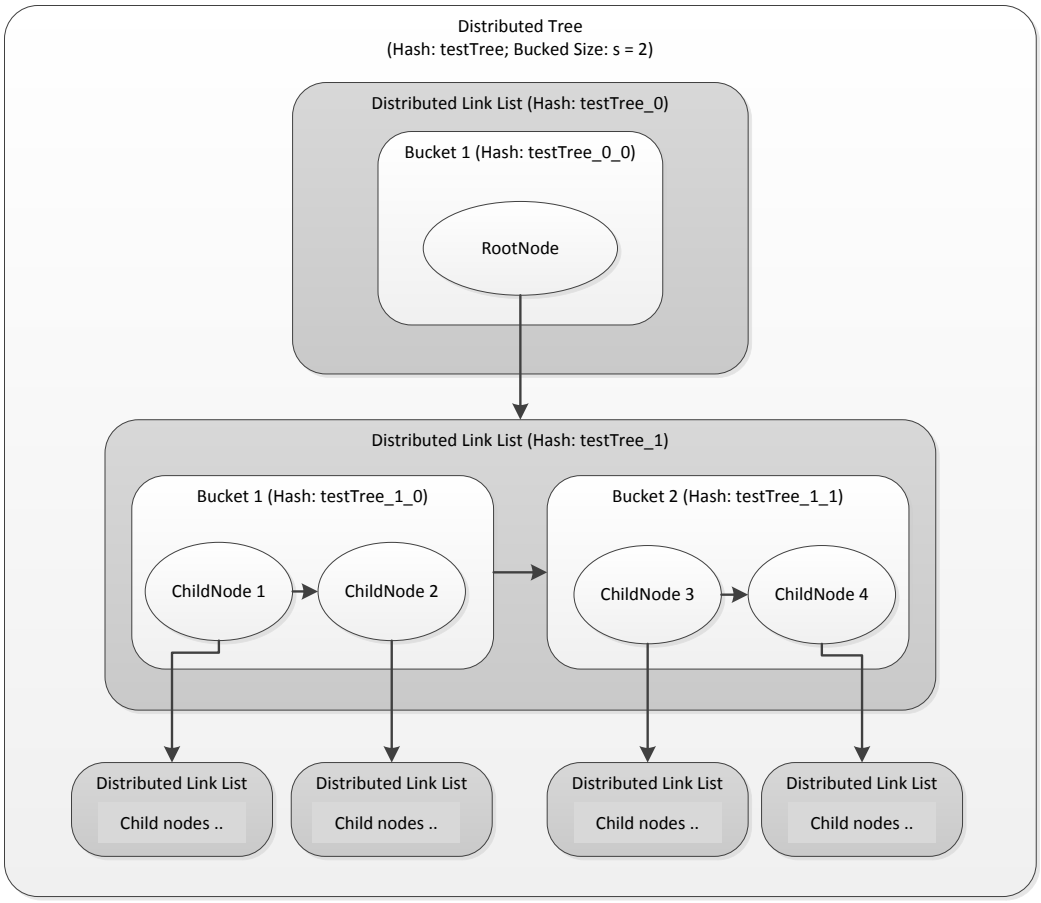


Figure 2.2: Distributed tree structure of bucket size(PG2012)

2.3.3 Plugins

Plugin layer is the layer above the framework, which provides various functionalities to the LifeSocial. Plugins are based on OSGi which makes the LifeSocial easier to extend the functionalities. Eclipse RCP is used for developing these OSGi based plugins. There are various plugins developed in the LifeSocial. Some of the important plugins are login, profile, message, wall, chat, friends, group, file, calender, search, audio/video chat, voting and notification plugins. In this thesis, The Groups plugin is extended to add more features to it.

2.3.4 GUI

This is the topmost layer in the architecture. This layer is responsible for developing user interface. GUI is displayed in a browser. The technology used in the GUI are HTML5, Ajax, Knockback, Bootstrap and Jquery. It also provides multi-

lingual support(English and German). The GUI is more user friendly and gives a feel good look. The plugin layer provides all the results to the GUI, and GUI is responsible for displaying it in the browser. The parts in GUI layer includes webprovider, plugin template and plugin logic. Plugin template is designed using te html file which uses knockout.js. Knockout.js is a javascript implementation, which uses Model-View-View-Model(MVVM) paradigm. Plugin logic is responsible for sending the user actions in the GUI to the WebProvider and receiving the results from the WebProvider and render it in the plugin template. As MVVM is used in the plugin template, Models, Views and Collections are used in plugin logic. These are created in the javascript file using knockback.js which is a combination of knockout.js and Backbone.js. WebProvider is the one that bridges the connection between the plugin and the GUI layer. WebProvider is developed as an OSGi plugin. This plugin is responsible for connecting all the API's of the plugin components to the frontend of the application.

3 Design Of Group-based Plugin

The chapter describes about the java implementation of the Group-based plugin. The Groups plugin which was previously developed has some limitations like creating one group and adding only friends to that group. This chapter gives an overview of extended group functionality by creating groups inside groups which is referred as community and the features included in the community. The first six sections are written by author Christy Jovita Bibiana Arockiaraj and the next 6 sections are written by Manasa Veena Rapolu.

3.1 Groups Plugin

This section gives an overview of the groups plugin and the functionality included in it to extend the groups plugin. The groups plugin is very restricted without any flexibility to add subgroup to main groups. After creating the groups, only friends of the owner can be added to the main group and the members that are added to the main group can create folders and upload files inside the group. Therefore, there is only one level hierarchical structure of groups. Therefore, adding members to the group is very restrictive in the old group structure. User can create a group and the group can have multiple folders and files in it. Further, every folders can have multiple folders and files in it.Figure 3.1 shows the structure of the old groups plugin.

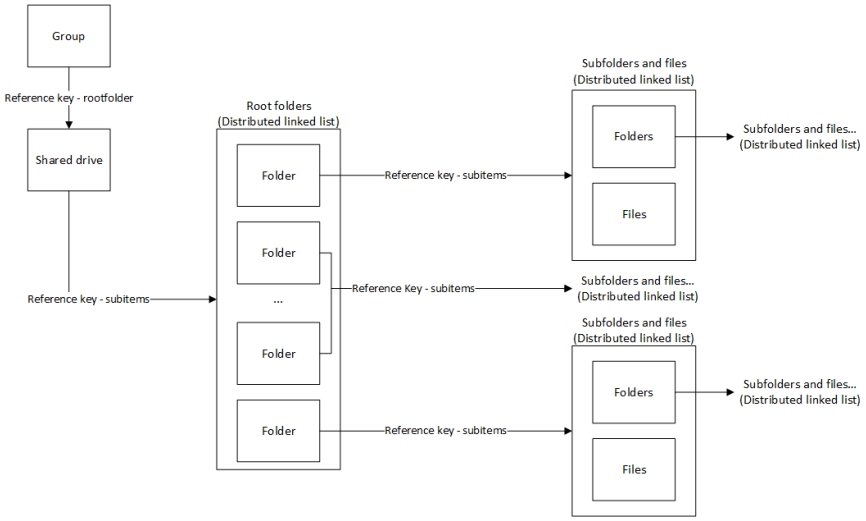


Figure 3.1: Old group structure [2]

To overcome these limitations of groups plugin, the functionality of it has been extended to create groups and subgroups with access controls. Therefore, the main goal of the thesis is to extend the group functionality by creating communities and inner level communities. Moreover, not only friends but random members can also be searched and added to the group. The members added to the group can create folder and files inside top community and the inner communities which is based on the read/write access permission. The rules to be followed when creating communities and adding members are as follows.

1. When new inner community is created, the child community should import all the Read/Write(RW) access users from the parent community and gives them also RW access rights to that inner community.
2. When adding new member to any community say ’A’
   - If RW access rights are given to the user, then provide the user with RW access rights to all the inner level communities that are created under community ’A’.
   - If only Read access is given to the user, then provide the user only read access right to all the parent communities up to the top level community.

Note that in the thesis, extended groups with additional functionality is referred as community. Figure 3.2 shows the structure of the community.

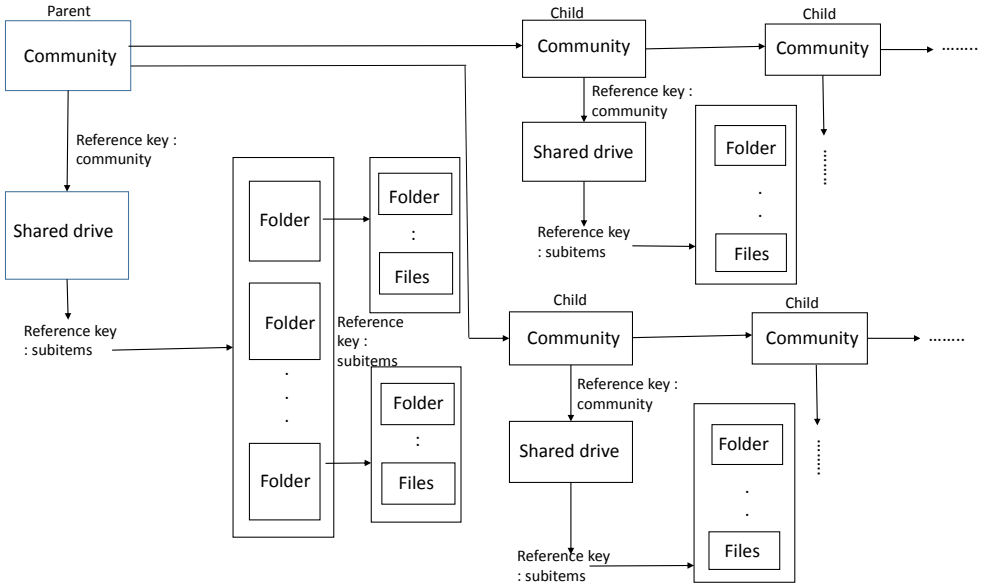


Figure 3.2: Community structure

The class GroupsPlugin is used for the creation and maintenance of both groups and communities. The main difference between groups and communities are the hierarchical structure and the read/write access of a member over each group/community. Therefore, during the creation of groups and communities, the reference is set to *"rootfolder"* and *"community"* respectively in the **Group** class using the method **setOneReferenceID(reference,ID)**, where ID is the identifier of the respective group/community being created. Every method in the **GroupsPlugin** class has an additional parameter as "reference" to distinguish between groups and communities.
The access control mechanism and the structure of the groups is simpler compared to that of community. In other words, community is a more sophisticated structuring of groups. Communities have inner level communities, whereas in groups there is no inner groups. The similarity between the two is the creation of folders and uploading/downloading files to them.

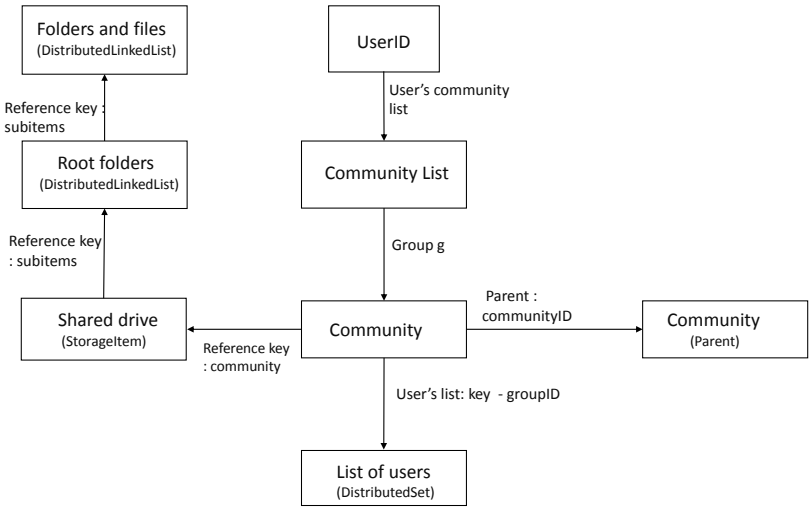


Figure 3.3: Data structures used and mapping of storage items in the community

The Figure 3.3 gives a brief overview about the data structures used in storing the created communities and its contents. User may be a member in more than one community. Every community has a list of users which is stored in **DistributedSet** using the key as communtiyID. Community is referenced using the reference key **"community"**. A community can be a parent community or in an inner level community. If community is created under a community, then parent is set for the community. The community contains rootfolder and folders and files which are retrieved using the **DistributedLinkedList** with the reference key as **"subitems"**. The upcoming sections describes about the creation of communities, adding members with read/write access, creation of folders and uploading/downloading files.

3.2 Create Community

Table 3.1: createPublicCommunity

Method name	createPublicCommunity
Description	To create top level communities
Input parameters	String communityname - name of the community String reference - reference for the community
Return type	Group

This method is responsible for creating the top level community, which does not have any parent. So when the community is being created in the constructor method of Group class, the parent is set to null. The owner of the community is set as the logged-in user. After the creation, the community is stored in the framework using **StorageDispatcher** by storing the community item along with the respective community key, which is generated using **getGroupStorageKey(community)** method. Further, the user list is stored as a **DistributedSet**.
There are two ways of mapping community and user. One way is to map the list of users that are added as a member to a particular community(*UsertoCommunityList*) and the other is to map the lists of community that the users are added as a member(*CommunityToUserList*). The first is done by fetching the users as DistributedSet using the key generated by **getGroupUserListStorageKey(community)** and mapping it to the community. The later is done by storing the list of community in the StorageDispatcher using the key as userID generated by **getUserGroupListStorageKey(UserID)** method.

3.3 Create Inner-level Communities

Table 3.2: createInnerCommunity

Method name	createInnerCommunity
Description	To create inner level communities
Input parameters	String communityname - name of the community String communityId - community identifier for the community String folderId - root folder identifier for the community
Return type	Group

This method is used to create the inner level communities in the hierarchy. When creating a community, reference ID is stored as "community" using **setOneReferenceID** in an abstract class **AbstractStorageItem**.
Each user is identified uniquely by userID which is generated by the framework. When a new community is created by the user, it is added to the list of communities of the user. When he tries to create a community inside already existing community, that existing community is searched in the list of communities of the user and is set as a parent community using *setParent(group)*. Users are then added to the community using **joinGroup(final Group group,**

UserId). In order to fetch the users from a particular community, **DistributeSet<UserID>** data structure is used by passing the community as a key to the method **getGroupUserListStorageKey(Group g)**.
Access controls are set for the communities by checking whether the community has a parent community or not. If the community does not have any parent, then all the users fetched from the DistributedSet are added to the *LinkedList<UserID> ACLList* and the write access is set by passing the ACLList to **setWriteAccess(LinkedList<UserID> writeAccess)**. If the community has a parent community, then the users having RW access rights of the parent are imported to the child community and is set using the method setWriteAccess() method. Therefore, the first rule mentioned in Section 3.1 is satisfied.

3.4 Add Folder

Table 3.3: addFolder

Method name	addFolder
Description	Create folders inside the community
Input parameters	String fName - name of the folder to be created String groupID - community identifier String FolderID - Root Folder of the community String reference -reference for the community
Return type	Boolean

This method is responsible for creation of the folder inside the community. It is always necessary to make sure that the user wants to create folder inside it is the selected community. Therefore, the method **changeGroup(groupID, reference)** is responsible for changing the current community to the user selected community. Further, the inner items of the selected community is retrieved and stored in a **GenericStorageItem<HashMap<String, Item»**, where the *String* is the folder identifier and the item is the folder object.
In order to get the parent reference of the item in **DistributedLinkedList(DLL)**, a workaround method **searchFolder(folderItem)** is employed to backtrack the DLL and to find the parent reference of the item.
Once the item reference is fetched, the folder is created using the **createFolder** method. The **createFolder** method is responsible for creating the folder and adding other properties to it like creating the access controls for the users and also to associate the folder created to the parent folder and parent community. Consequently, the folder is added as a inner item to the parent

folder/community.

3.5 Leave Community

Table 3.4: leaveGroup

Method name	leaveGroup
Description	To leave the community
Input parameters	String groupID - community identifier String reference -reference for the community
Return type	Boolean

This method is responsible for leaving the community by the user. User can leave the community and no more able to have access to the inner items of the community as well. When leaving the community, all the mapping between the user and the community has to be removed. Therefore, the initial step is to remove the user from the mapping of *CommunityToUserList*, in which the user is removed from the DistributedSet<UserID> of the community. Next step is remove the community from the *UsertoCommunityList* using the callback function **getMyGroups**.

3.6 Access Control Mechanism

Access controls like read/write access are given to the users. It is important to set the access rights to the user when he/she is added to a particular community. Therefore, the access rights are set to the users in various places like 1) when new community is created and 2) when user is added to the community. The first option invloves the user automatically gets read/write access rights to the created community. The second option involves searching users and adding explicitly read/write access to the users. If the user is given the read access, then the user can read the current community as well as parent communites. If the user is give write access to the community, the user can have the freedom to create community or create folder or upload files in the current community. The users with the write access are stored in **LinkedList<UserID>** and is set to the appropriate community using **setWriteAccess(LinkedList<UserID>)** method and then it is stored in the respective community using **storeItem()** method. The access rights are set in the createPublicCommunity, createInnerCommunity, createFolder and addFriendstoCommunity.

3.7 Adding Members To Community

Table 3.5: addFriendstoCommunity

Method name	addFriendstoCommunity
Description	Adding members to the community
Input parameters	String GroupID - community identifier for the community String[] friends - List of friends boolean isWriteAccess - To check for access rights String reference -reference for the community
Return type	Boolean

The method is responsible for adding members or friends to the selected community. The input parameter *String[] friends* has list of members name to be added to the community. The *boolean isWriteAccess* contains the boolean value which determines the access rights for the user.

The user ID for each members are fetched by passing the user name as an input to the method **getUserIDfromName(string)** of the UserManager class. Later, every user ID is added as a member to the community by **joinGroup(temp, id)**, where temp is the community. The joinGroup() method uses the **getUserGroupListStorageKey(id)**, where id is the userID which is used as a key to store the community in StorageDispatcher. Therefore, the communities are stored for a user by **storeItem(storageItem, getUserGroupListStorageKey(id)**, where storageItem is the community and later it can be fetched using the getInnerCommunityList() method or getCommunityList() method.

Furthermore, based on the boolean value isWriteAccess, the user is added to the LinkedList<UserID> and set in setWriteAccess(ACLList) method. Consequently, the community is stored in the framework.

Finally, the notifications are sent to the corresponding community member that the user has been added to the community successfully.

3.8 Deleting An Item

Table 3.6: getContentsForCommunity

Method name	deleteContent
Description	Delete the contents of the selected community
Input parameters	String parentItemId - community identifier for the community String itemId - item identifier for the community String groupID - community identifier for the community String reference -reference for the community
Return type	-

To delete a particular item in the community, it is necessary to pass the parent ID of the item, its own ID and the ID of the community that it belongs. The first step is to retrieve the contents of the selected community and search for the item reference in the DistributedLinkedList. The second step is to delete the item from the **groupContentMap** and consequently it should be notified to all the other depending objects of the deleted item. Finally, the selected community is stored after the deletion of one of its child item.

3.9 Renaming Community

Table 3.7: renameCommunity

Method name	renameCommunity
Description	To rename the top level and inner level communities
Input parameters	String communityname - new name for the community String communityId - community identifier for the community String reference - reference for the community
Return type	-

The method is used to rename the community. The input parameters are the current community identifier and the new name. The changeGroup() method returns the currently selected community object with the input community

identifier. Further, name of the selected community is changed to the new name using the method **setName(String name)**. Later, the selected community is stored in framework.

3.10 Uploading/downloading Files

Table 3.8: Uploading files

Method name	addFile
Description	Add the file to the community
Input parameters	LinkedList<FileInfo> files - List of files String groupID - Community identifier String FolderID - Folder Identifier String reference -reference for the community
Return type	List<File>

This method is responsible for adding the files to the folder or the community. It is always important to make sure that the user is currently in the selected community using changeGroup() method. In a similar way as folder, inner content of the community should be fetched and then the parent reference of the item should be set. Later the files are added to the community using the method **addFile(item, selectedgroup, fi)**, where item is the folder, selectedGroup is the current community, and fi is the file to be added to the community. Finally, the added list of files is returned. Downloading the file is done by **getFileToOutputStream()** method in **FileStoragePlugin**.

3.11 Get inner level communities

Table 3.9: getInnerCommunityList

Method name	getInnerCommunityList
Description	Fetches the inner level community
Input parameters	String reference -reference for the community
Return type	List<Group>

This method is responsible for fetching all the inner level communities of the user. In order to fetch the inner community of the user, the entire community

list of the user is fetched. The list of user's community is fetched using the callback function **getUsersGroups**, in which all the communities are fetched and checked for the parent value. The list of community is retrieved as a **GenericStorageItem<HashSet<UUID»** using the key value **getUserGroupListStorageKey(uid)**, where uid is the user identifier.
If the parent of the community is not null, then the community is added to the *innerGroups* list. Finally the list of community is returned.

3.12 Get Contents Of A Community

Table 3.10: getContentsForCommunity

Method name	getContentsForCommunity
Description	Fetches the contents of the selected communitys
Input parameters	String communityId - community identifier for the community String folderId - folder identifier for the community String reference -reference for the community
Return type	GroupContentDescription

The methods allows the user to fetch the contents of the current community. When the user enters into a community, the contents inside the community are retrieved and presented. There are two cases that the user can enter into a community.

1. User enters into the top level community for the first time.
2. User currently in an inner level community and wants to switch to another community.

If the user is in other community and switches to another community, the selectedGroup is changed using the **changeGroup** method. The changeGroup method checks whether the user enters the top level community, in other words the selectedGroup is null, otherwise it changes the selectedGroup to the currently selected group. The changeGroup method also fetch the users in the community. The users are stored in DistributedSet and are stored in the framework using the key **getGroupUserListStorageKey(selectedGroup)**. The contents retrieved should be checked for the access rights using the **checkForWriteAccess** method. Moreover, if the user logged-in is the owner of the item, the setUpdateAllowed method grants the logged-in user to rename or delete the item. The contents in the community are of three categories.

1. Community
2. Folder
3. Files

The inner communities are retrieved using the **getInnerGroupsList()**, which fetches all the inner communities of the user and later the retrieved communities are checked whether they are the children of the selected community. The folder and files are stored in the *DistributedLinkedList* and are retrieved using the *getSubItems* method. Later the contents are then added to the Linked list of type AbstractStorageItem, which contains all the inner contents of the community.The hierarchy of the community and the inner contents of the selected community are maintained in a list "LinkedList<AbstractStorageItem> folderPath" which is set in *set_breadcrumInformation*. There are three different hierarchical structure in designing the inner contents of the community:

1. Community is created inside a community
2. Folder is created inside a community
3. Folder is created inside a folder
4. Files inside a folder

The *set_breadcrumInformation* keeps track of the hierarchy of the community structure. To conclude, this methods returns the **GroupContentDescription** which contains the members of the community, inner community, folders, files and breadcrumb information to navigate to the upper hierarchy levels.

4 Design Of Graphical User Interface

The chapter gives an overview of the designs of each functionality included in the plugin. The GUI layer is responsible for displaying the results from the java backend implementation. It is based on MVVM pattern. The technology used are knockback.js, JQuery, AJAX, HTML5, css and Twitter bootstrap. WebProviderPlugin bridges the connection between the Plugins and the frontend design. Figure 4.1 shows the connection between the WebProviderPlugin which consists of **CommunityPluginHandler** and **GroupPluginHandler** and the File system used and Java API calls. CommunityPluginHandler is responsible for sending the request to java API and receiving the results from the plugins. RESTHandler class in WebProviderPlugin is responsible for java API calls by initializing the CommunityPluginHandler class. It is responsible for handling all the calls to the respective plugins. The results from the plugin is serialized to JSON object and is send to the javascript file and later it is presented in the html for the user view.

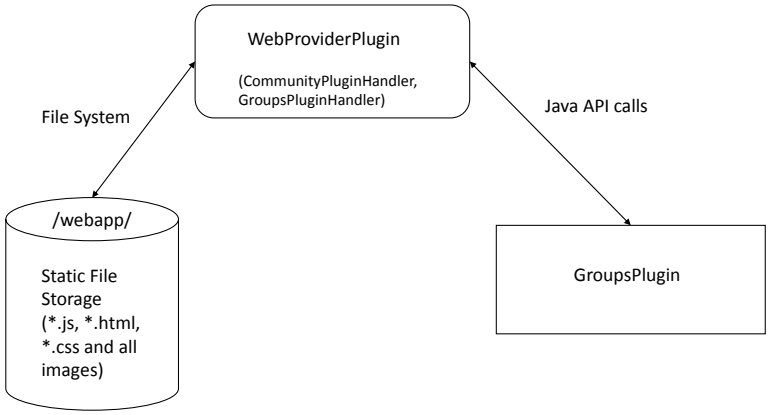


Figure 4.1: WebProvider plugin works in between File system and Java API[2]

All the static files are in the webapp folder inside WebProviderPlugin which consists of files like

1. HTML

2. css
3. Javascript
4. Images

These files are responsible for the design of the GUI. The upcoming sections gives a overview of the design of each functionality in the community plugin. For the extended Groups plugin, a new tab "Community" is introduced in the LifeSocial page.

4.1 Design of Top-level Communities

Top-level communities are the higher level communities in the hierarchical structure. These top-level communities have no parent community. The first page of the community consists of list of top-level communities. Each communities listed in the first page are created as a url link to navigate inside the selected community. Every community has its own *community* identifier and *rootfolder* identifier. Once the link of the selected community is selected, it is navigated to the next page of the selected community. The Figure 5.3 shows how the page displays the list of communities.

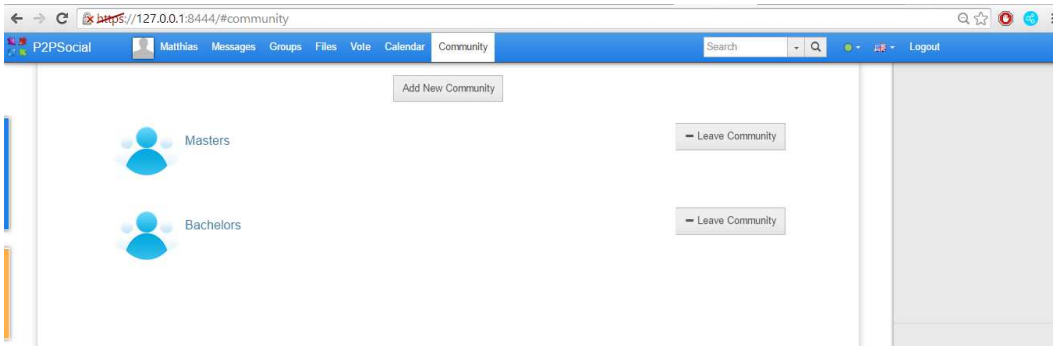


Figure 4.2: Display of top communities

4.2 Design of Inner-level Communities

Inner level communities are the communities that are created as a child community to the top level community. After clicking on the link of the top level community, it navigates to the inner page, where the list of inner communities are displayed in the left section of the GUI. The left section of the GUI is divided into two parts. The first section is to display the name of the current selected community, which is a link to the home page of the current community. In the

first part, next to the display of name of the community, a drop down arrow is placed. The drop down arrow when clicked displays a drop down box which has input fields and buttons to create inner communities, folders, renaming the community and uploading the files. The Figure 4.3 shows the display options for creating communities, folders, renaming communities and uploading files

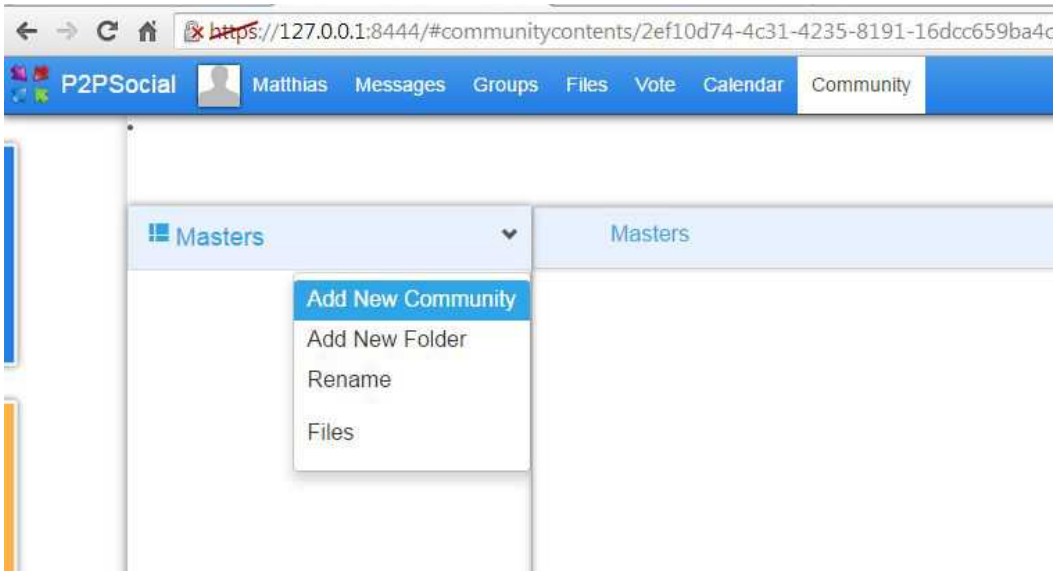


Figure 4.3: Dropdown list for creating inner communities, folders and uploading files

The second section displays list of all the child communities which are placed right below the first section. The Figure 4.4 shows the display of the list of inner communities.

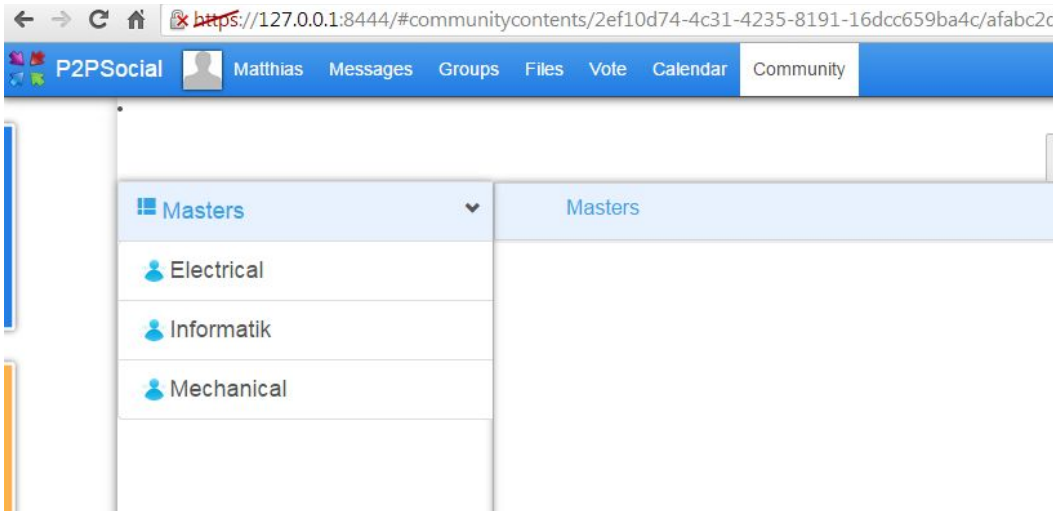


Figure 4.4: Display of inner communties

4.3 Navigation Through Breadcrumbs

The navigation to a particular community is done through breadcrumbs. Breadcrumbs makes the user to easily navigate through the hierarchical structure of the created communities. Breadcrumbs are present in the right section of the GUI. It helps user to keep track of the communities/folders within the hierarchical structure of the created community.

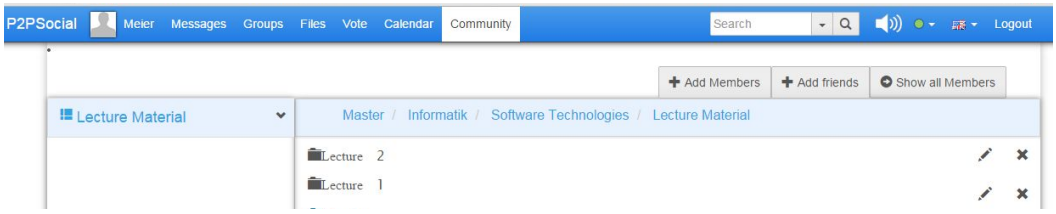


Figure 4.5: Navigation through breadcrumbs

4.4 Navigation To Inner Contents of Hierarchy

The inner contents of the community can be accessed by clicking on the communities/folder displayed in the left section of the GUI. As mentioned in the section 2, the the left section's first part has always the currently selected community name, which makes the user easier to navigate to the community, if they are currently in any of the inner folder.

4.5 Adding Members

For every community, friends are added using a modal dialog box. The modal dialog box consists of an input field to search the name of the user connected in the network. The user can type the name of the user to be added as a member to the community. After entering the name of the user followed by a click on the search icon, the framework displays the list of users similar to the user name given in the search field. The list of users are displayed sequentially in the dialog box. The user can select the list of users individually to add. In the footer of the modal dialog box, it consists of a check box to select for read/write access. If the user wants to give access rights to the searched list of users selected, then the check box can be selected. Otherwise it can be left unchecked. Then the selected users can be added to the community by clicking on the OK button in the modal dialog box. Figure 5.11 shows the search results of the searched members and the members are selected with write access.

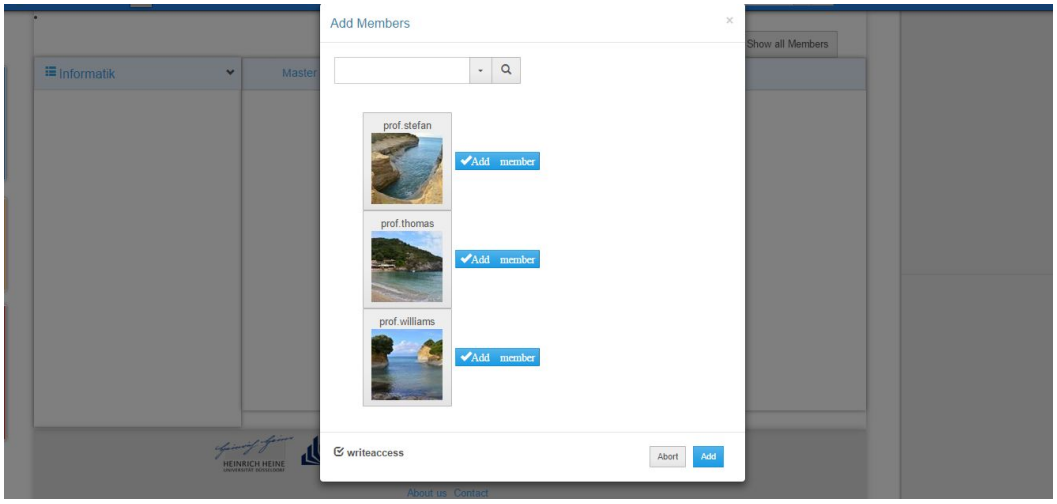


Figure 4.6: Adding Members to the community

4.6 Adding Friends

The friends of the user can be added as a member to the community by clicking on the **Add Friends** button. the button opens a modal dialog box which displays the list of friends. From the list of users displayed in the modal dialog box, the user can select all or few of the friends to add them as a member to the community. Figure 5.27 shows the display of friends and the checkbox to select the friends to add as a member to the community.

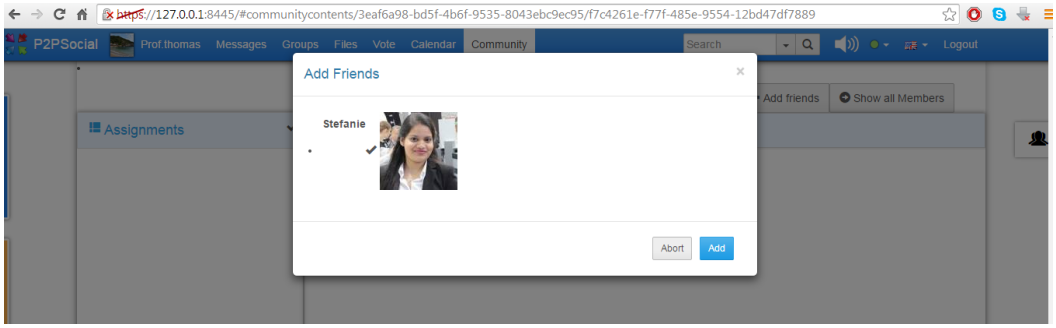


Figure 4.7: Adding friends to the community

4.7 Display Members

The members are displayed by clicking on the **Show All members** button in the top right corner of the page. It displays the members of the current community in a modal dialog box. Figure 5.12 displays the members of the current community in the dialog box.

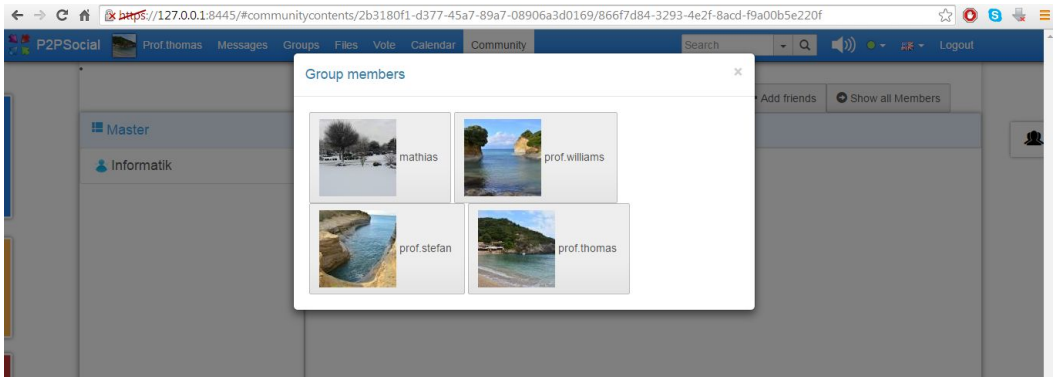


Figure 4.8: Display members of the community

4.8 Upload/Download Files

Files can be uploaded and downloaded in the communities. To upload a file, one should select the **Files** options from the drop down menu by clicking on the drop down arrow which is present adjacent to the community header in the left section of the page which is shown in Figure 4.3. When it is clicked, the file selector dialog opens up allowing the user to choose the file from their drive. Once the particular file is selected, it gets uploaded to the community and is displayed in the right side section of the page. Files can be downloaded

by simply clicking on the file names. Files can be deleted based on the access rights to the particular user. Figure 5.28 shows the list of files in the folder displayed in the right section of the page.

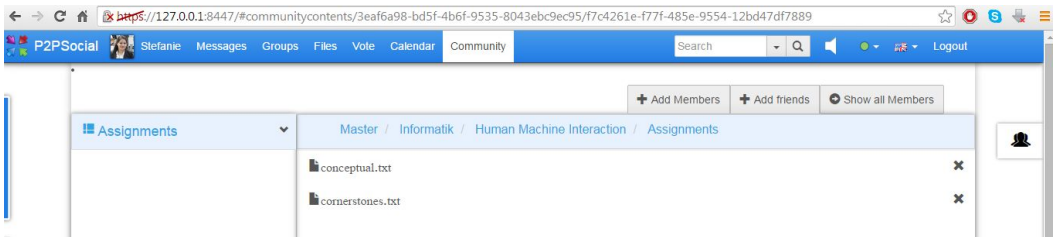


Figure 4.9: Downloading files

4.9 Delete

Delete function is used to delete the folders and files completely from the newtwork. As the folders are displayed on the right hand side of the page, a delete button is seen with *glyphicon-remove* symbol which is placed to the right end corner of each folders and files. This *glyphicon-remove* should be clicked in order to delete that particular folder. When clicked on delete, a modal dialog box opens up asking whether the user really wants to delete the community. User can confirm the deletion by clicking on delete which is present in the footer of the modal box or can click cancel to revert back the operation. Figure 5.20 shows the modal box for deleting folder or files for confirmation.

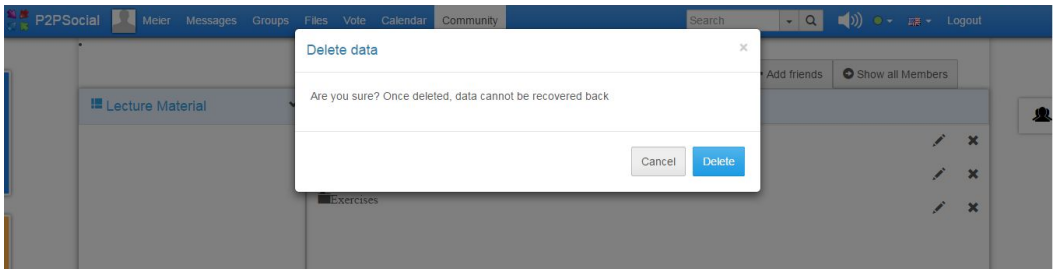


Figure 4.10: Deletion of Folders and Files

4.10 Rename

Rename function is used to rename the communities and folders. Communities can be renamed by selecting the **Rename** option from the drop down menu which is shown in Figure 4.3. When clicked on rename, a modal dialog box

opens up asking to enter a new name which needs to be given. After entering the new name for the community, it is confirmed by clicking on the rename button present in the footer of the modal box. In a similar manner, folders can also be renamed. As the folders are displayed on the right side of the page, a rename button is seen with *glyphicon-pencil* symbol which is placed to the right end corner. This *glyphicon-pencil* is clicked for renaming the folder. A modal dialog box is opened in a similar fashion to that in communities asking for a new name to enter for the selected folder. After entering the new name, it is confirmed by clicking the rename button from the footer of the modal box. Figure 5.22 shows the design of modal dialog box for renaming community and folder.

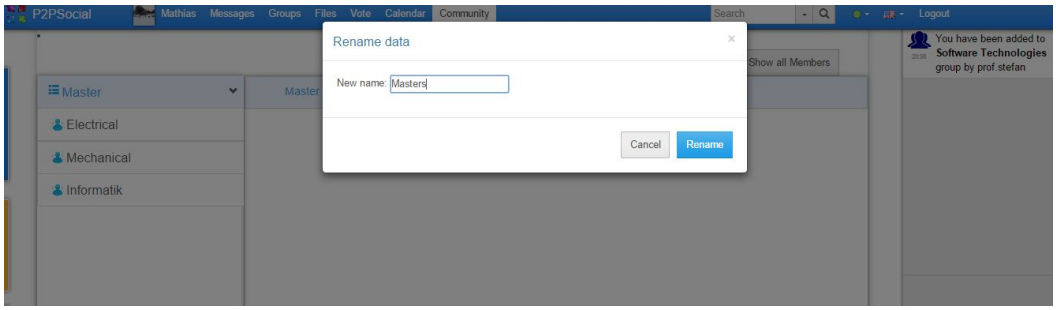


Figure 4.11: Modal dialog box for renaming Community and Folder

4.11 Leave Community

Communities can be left by the users by clicking on the **Leave community** button in the top page of the community. The user can click on the button to leave the community or abort the action. Figure 4.12 shows the the drop down box options to leave the community or abort the action.

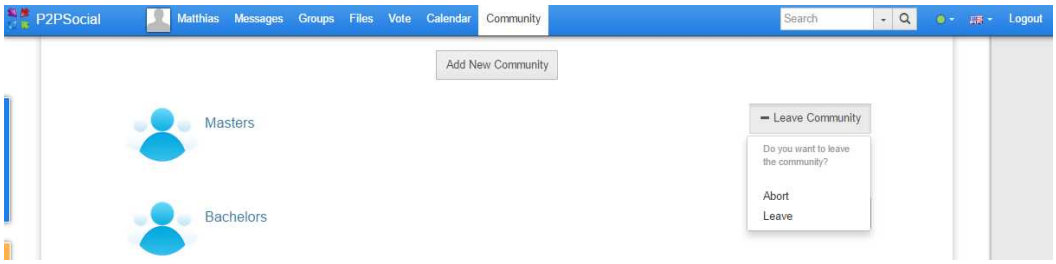


Figure 4.12: Dropdown box for leaving community

The chapter discussed about the html designs of the functionality included in the community page. The functionality like creating community, folder, renam-

ing and deleting uses modal dialog box as it gives the user a flexibility to enter details in a form based GUI. The icons used for deleting, renaming and showing the lists are taken from twitter bootstrap glyphicons. Therefore, the GUI design can be modified or added with more functionality to give the user good aesthetics.

5 Final Testing

This chapter describes about the testing that has been performed in order to test all the functionalities that have been implemented by both the authors. A test case scenario that is based on the university has been developed by considering users with different designations at the university as professors, assistants and students. The performance of the entire test plan along with the screen shots are described below. This chapter is written is by the author Manasa Veena Rapolu.

5.1 Test Plan

The test plan includes the set up of LifeSocial on total 10 machines in the pool area of the university. 10 different users are logged in with different designations. Like, 1 user is logged in as an administrator who is responsible for creating the main communities. 3 users are logged in as professors, 3 users as the assistants who are working under the professors and who are responsible for managing all the assignments and materials and 3 users as the students of the university. The test plan for creating communities is based on a use case scenario of university where it is used as a medium of communication among students and professors. The top-level communities would be the Masters and Bachelors courses. Inside Masters, different communities like Informatik, Electrical, Mechanical are created. Inside Informatik, several inner communities of different courses are created. They are for example, software technologies, embedded systems, human machine interaction. All these communities contain the course material arranged in different folders and files. The entire test plan scenario is shown in Figure 5.1

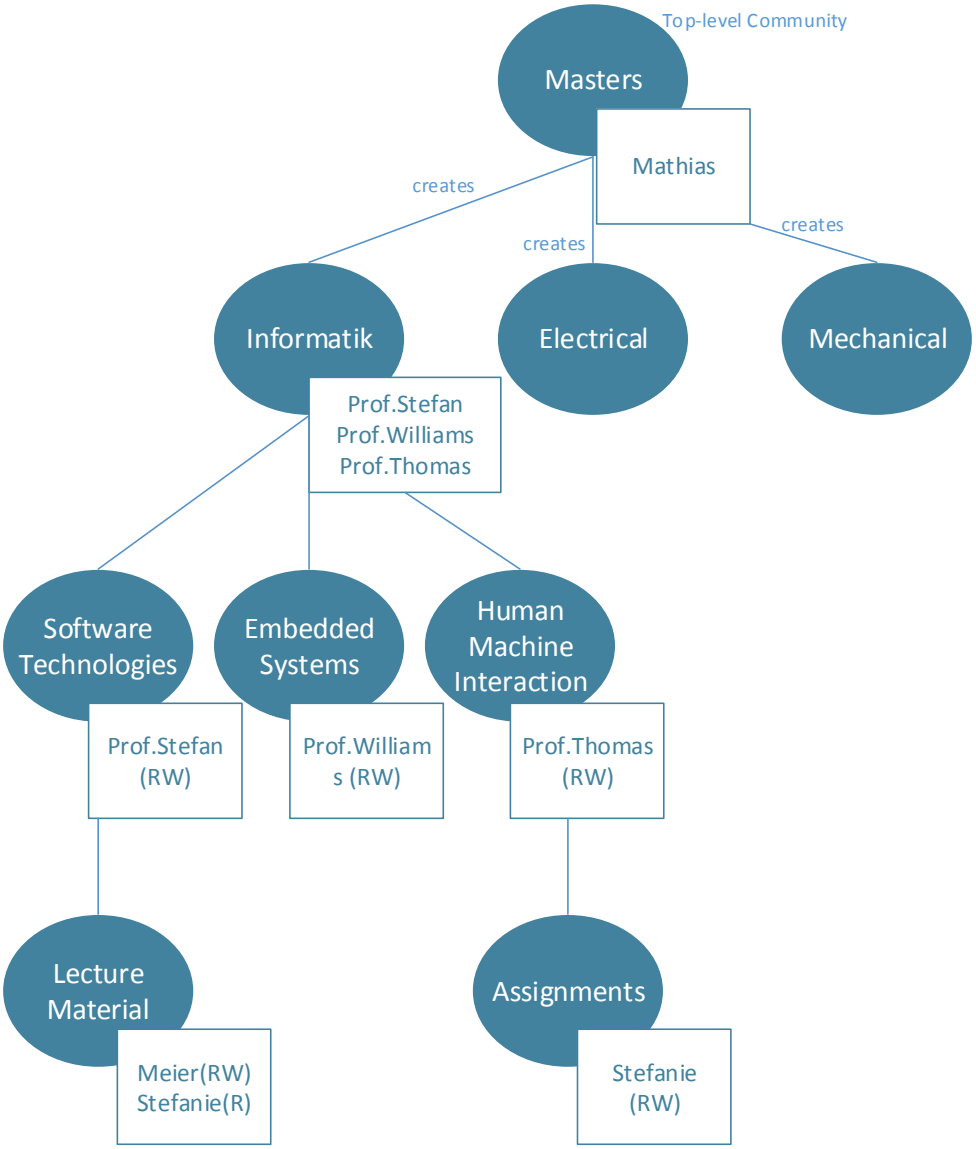


Figure 5.1: Test Plan Scenario

Firstly, the administrator named as Mathias creates the top level communities named Masters and Bachelors.

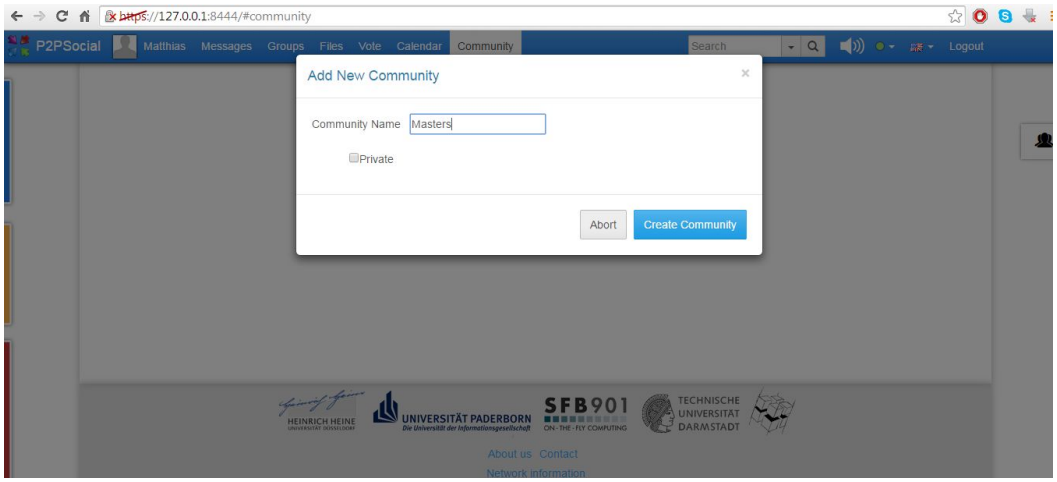


Figure 5.2: Creating Top-level Community

The top level communities created are shown as in the Figure 5.3:

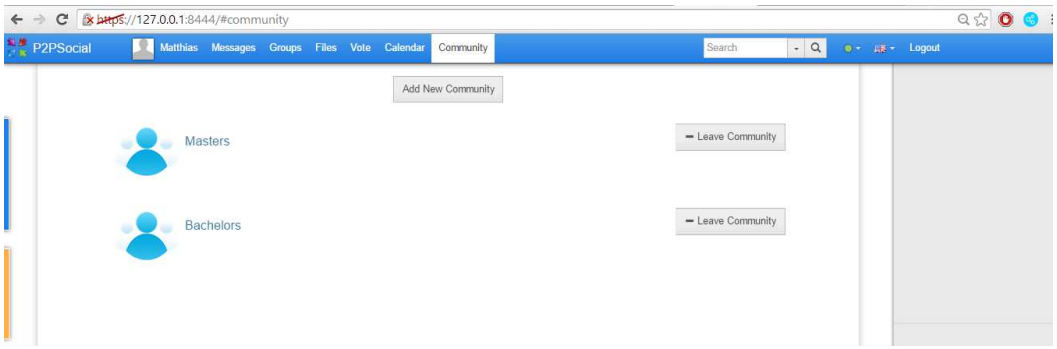


Figure 5.3: Top-level Communities

He then creates an inner community named Informatik inside Masters by selecting Add New Community from the drop down menu as shown in the Figure 5.4 and enters the name for the community as Informatik as shown in Figure 5.5.

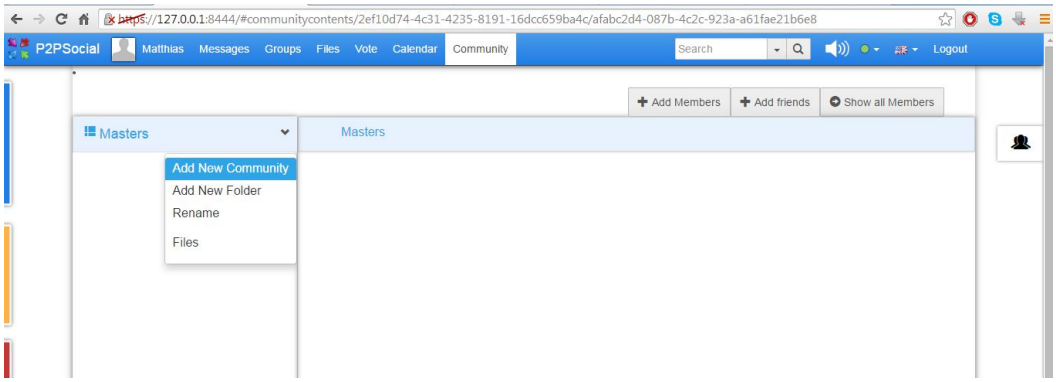


Figure 5.4: Creating Inner-level community

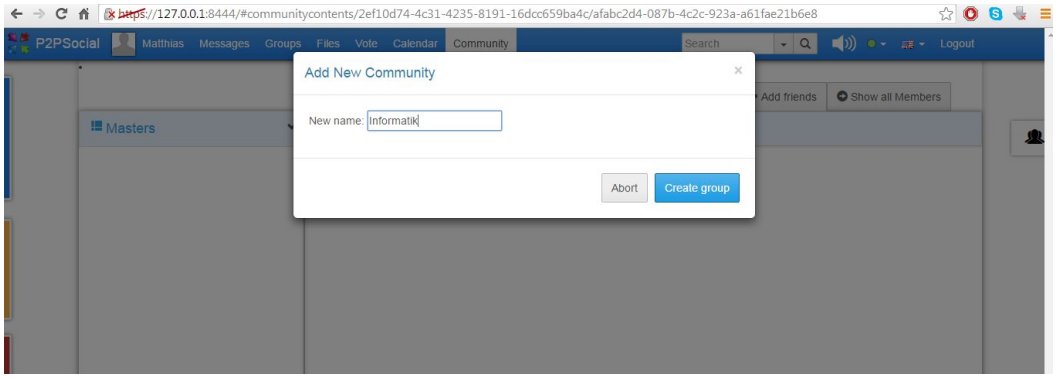


Figure 5.5: Inner Community

In a similar way, he adds two other inner communities named Electrical and Mechanical as shown in the Figure 5.6.

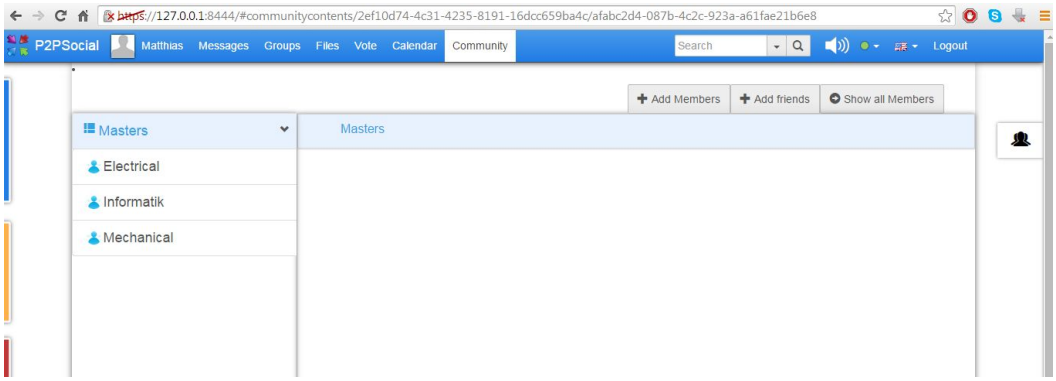


Figure 5.6: List of Inner Communities

Members can be added to the communities by clicking on the Add member button or by making them friends and inviting to communitites.
The professors who are responsible for the three departments (Informatik, Electrical, Mechanical) are added to the respective communities. For example, Prof.Stefan who belongs to Informatik department is added to the Informatik community by Mathias by clicking on the Add member button which opens up the dialog as shown in the Figure 5.7 .

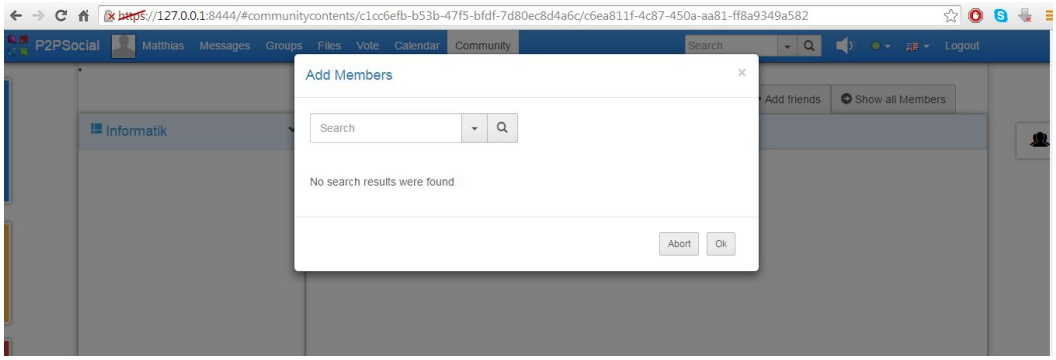


Figure 5.7: Add Members Dialog

The person to be added is searched for using the search drop down as shown in Figure 5.8. Here, Prof.Stefan is searched and the result is displayed as shown in the Figure 5.9.

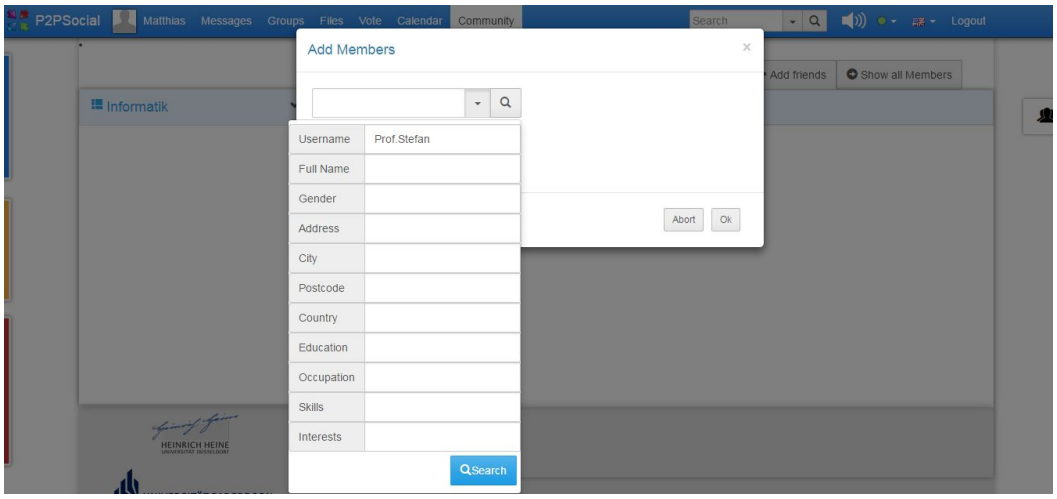


Figure 5.8: Search Dropdown

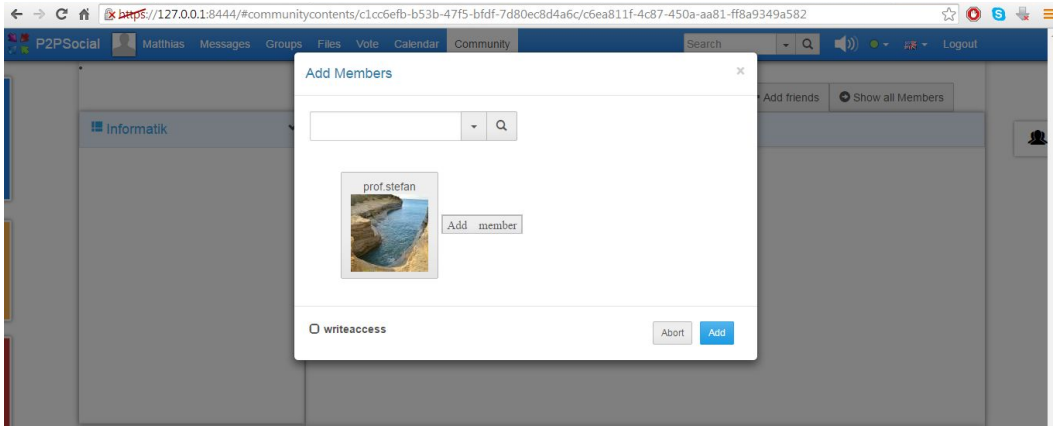


Figure 5.9: Search Result

After the results are displayed, the member is selected by checking the Add member checkbox and the writeaccess in the footer is checked if one intends to grant write access to the member or is left unchecked if only read access is to be granted. Here, write access is granted to Prof.Stefan by selecting him and checking the writeaccess checkbox in the dialog as shown in the Figure 5.10.

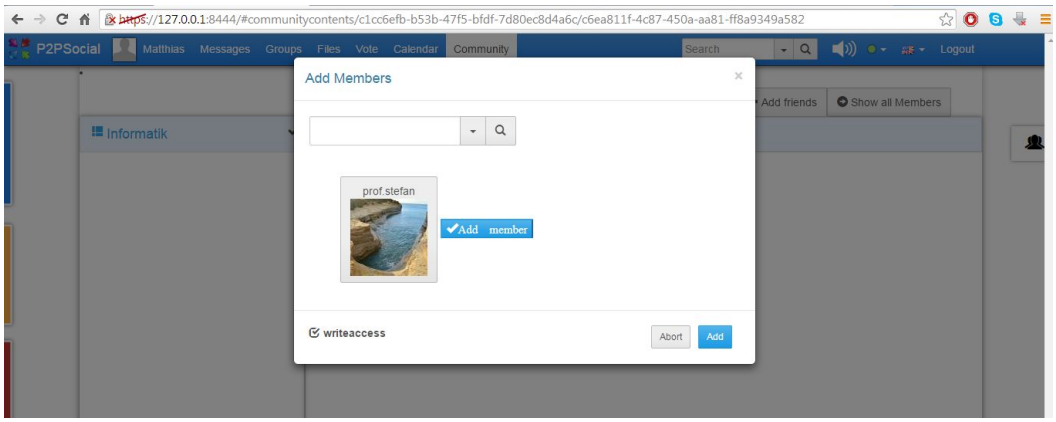


Figure 5.10: Adds Member To Community

In a similar fashion, he can add all the professors who are part of the Informatik department. He adds Prof.Williams and Prof.Thomas to the Informatik community as shown in fig. as shown in 5.11

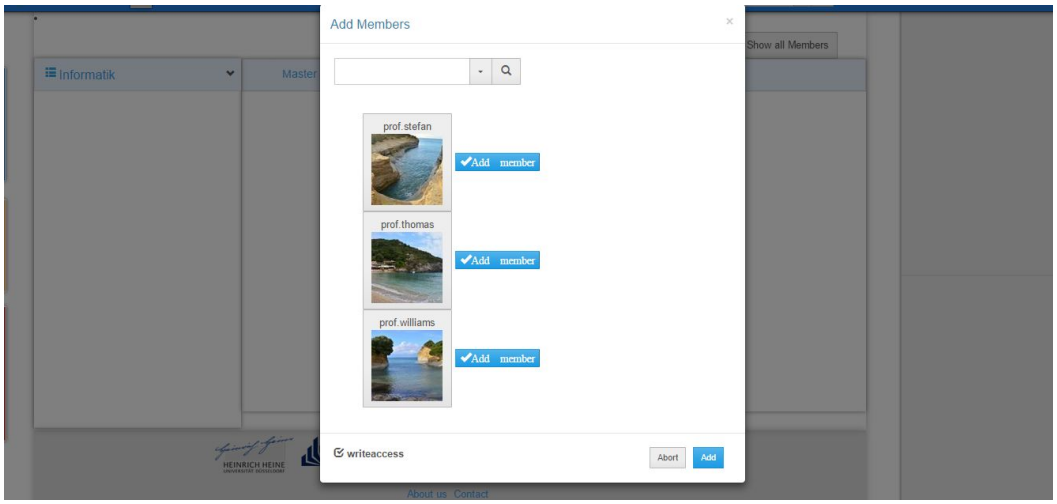


Figure 5.11: Add All Members

All the members of the Informatik community can be seen by clicking on the button Show All Members which shows all the persons who are part of this community. This can be seen in the Figure 5.12.

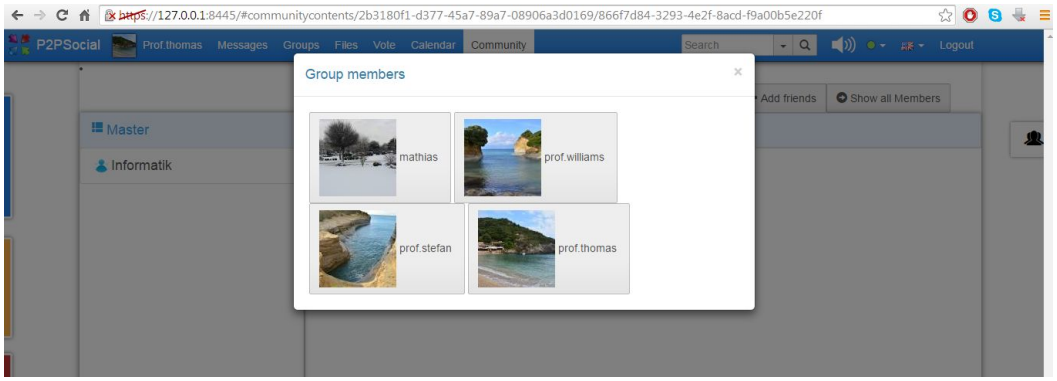


Figure 5.12: Show All Members Dialog

Once Prof.Stefan is added to the Informatik community, he receives a notification as shown in Figure 5.13 saying that he is added to the Masters community and Informatik community as a person added to inner community is directly added to its parent communities. Also, Prof.Williams receives the notifications in a similar manner as seen in Figure 5.14.

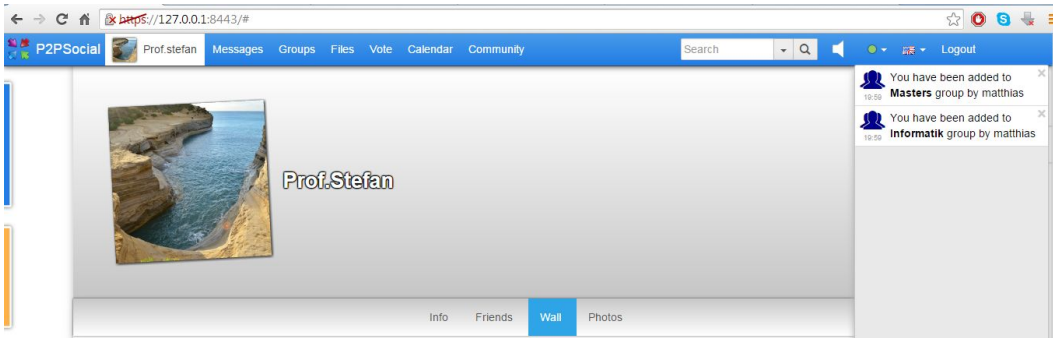


Figure 5.13: Prof.Stefan's Notifications

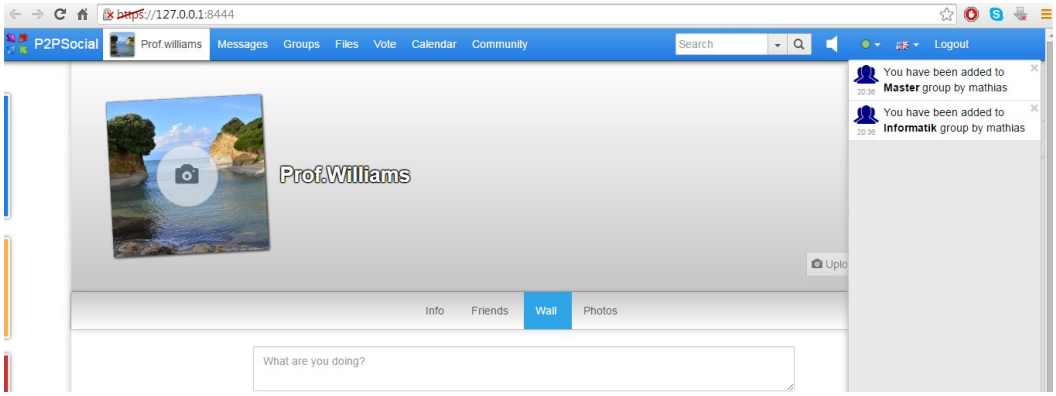


Figure 5.14: Prof. William's Notifications

Prof.Stefan can now see the Masters community in his community page and can navigate to Informatik community inside to which he is granted write access. His page looks as shown in the Figure 5.15.

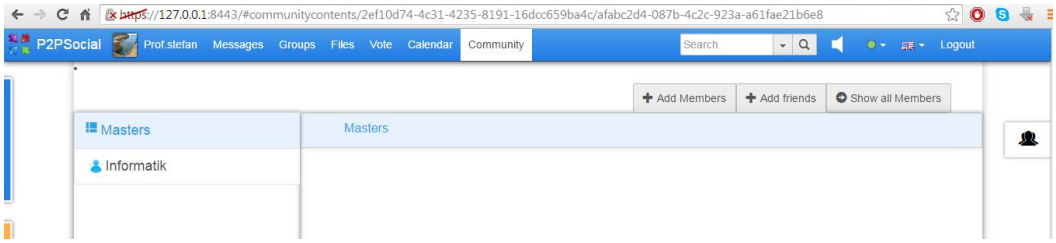


Figure 5.15: Inner Community of Prof. Stefan

Since he is granted with write access, he can see the drop down menu showing options of adding new community, adding new folder and files as shown in Figure 5.16. For a person having read access, this option is disabled. This means that the person having read access cannot create new community/folder/file.

Instead, he can only view the content.

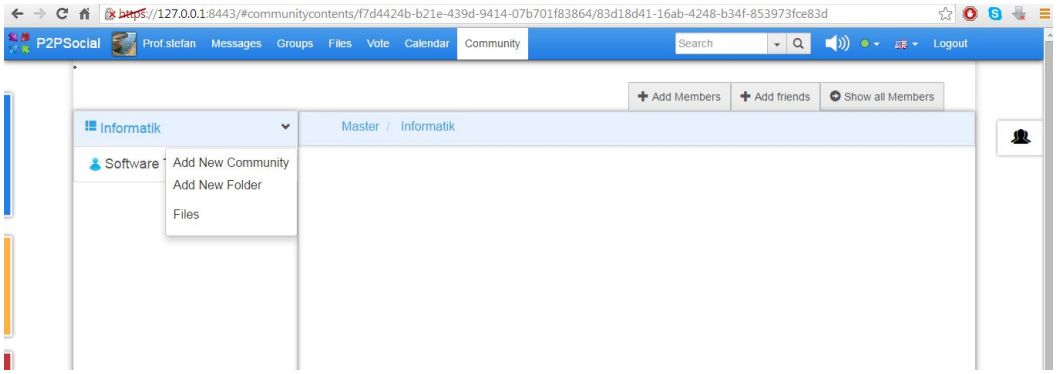


Figure 5.16: Prof. Stefan's Access Rights

Prof.Stefan now creates a new community inside informatik named Software Technologies to which he adds his assistant, Meier and grants him the write access. Meier is responsible for managing all the contents of Software Technologies. Meier's community page looks like in the Figure 5.17.

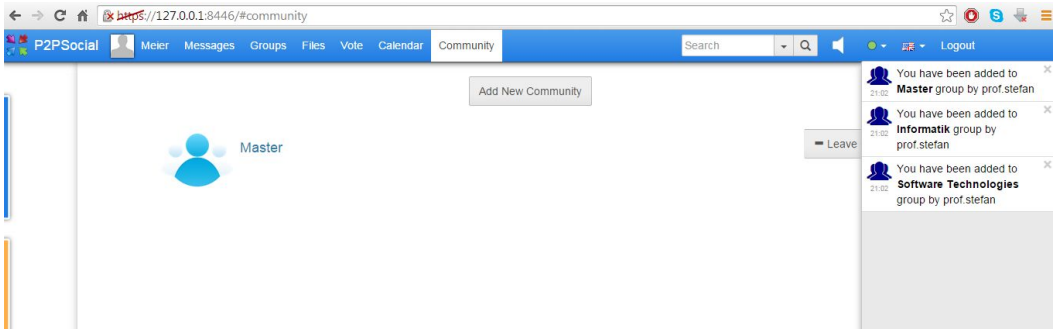


Figure 5.17: Meier's Community Page

Meier, having the write access creates a new community named Lecture Material in which all the lecture materials are uploaded. He then creates a first folder in this community named Lecture 1 by clicking on the Add New Folder option from the drop down that opens up a new modal to enter the folder name and add as shown in Figure 5.18.

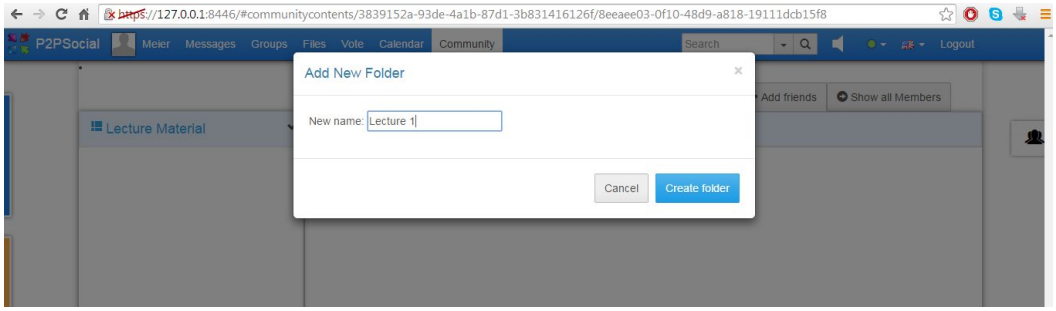


Figure 5.18: Creating New Folder

In a similar way, Meier creates several folders and the list of folders inside the community are seen as in Figure 5.19. Since, he is the owner of all the folders created, he has the rights to rename them and delete them by clicking on the pencil and cross symbol glyphicons to perform the actions respectively.

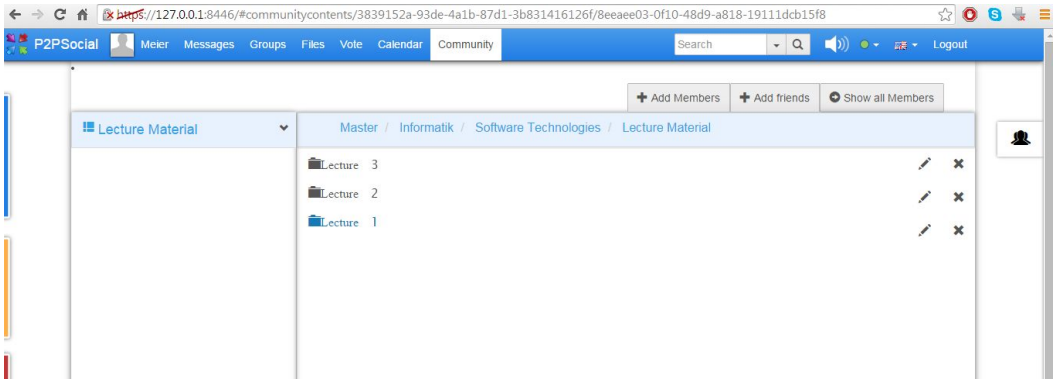


Figure 5.19: List of Folders Inside Community

In order to delete a folder/file, cross glyphicon is clicked which opens up a dialog asking to confirm the intended operation as shown in the Figure 5.20.

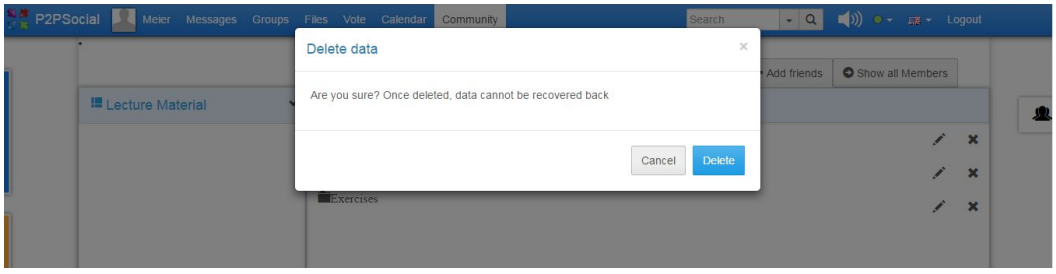


Figure 5.20: Delete Folder/File

Meier tries to rename a folder Lecture 3 to Exercises by clicking on the pencil glyphicon which opens up a dialog asking to enter the new name. The new name Exercises is entered as in Figure 5.21.

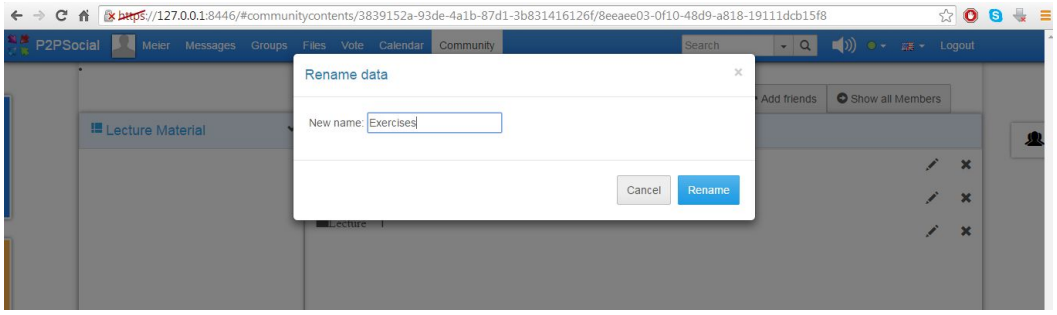


Figure 5.21: Rename Folder

The newly named folder Exercises is seen in place of Lecture 3 as in Figure 5.23.
Also, renaming a community is possible. This can be achieved by selecting **Rename** option from the drop down menu present beside the community name in the top left corner of the community page. The Figure 5.22 shows Mathias renaming the top main community from Master to Masters.

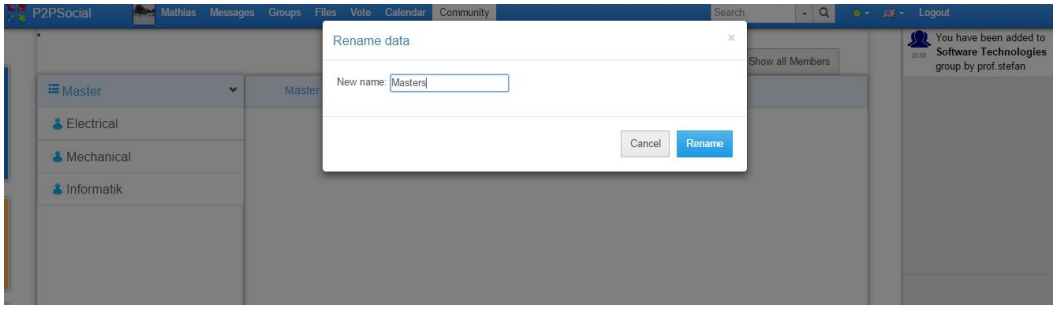


Figure 5.22: Rename Community

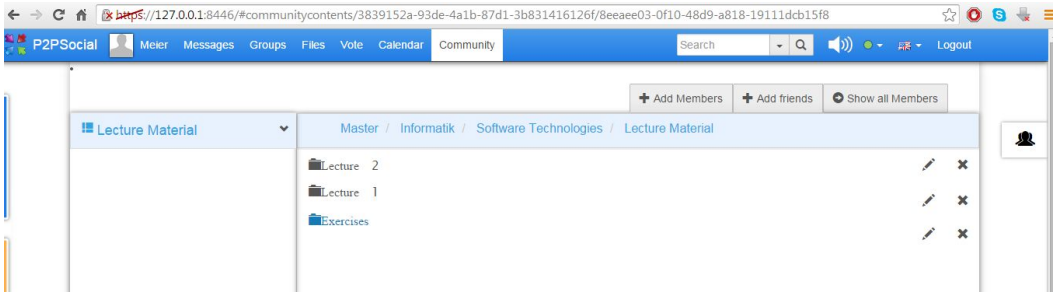


Figure 5.23: After Rename

Meier then adds a student named Stefanie to this community with read access. This student is automatically added to all its parent communities and receives the notifications as shown in Figure 5.24.

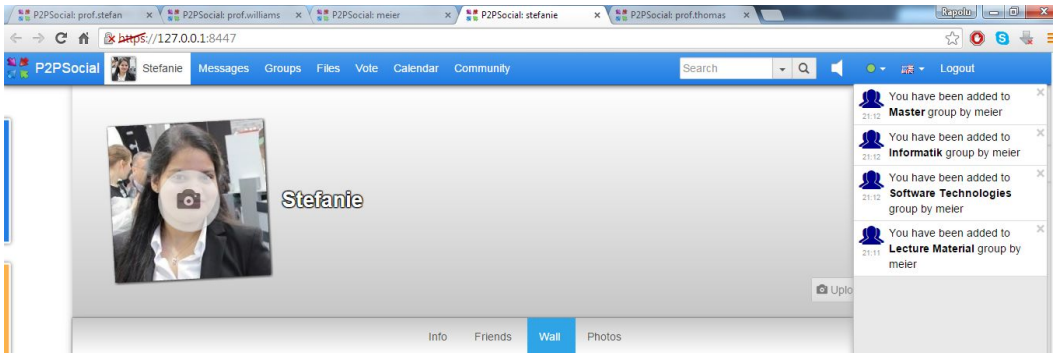


Figure 5.24: Stefanie's Community Page

This student can only read the contents from the folders but cannot rename and delete them. Permissions are not granted to neither rename or delete by disabling the glyphicons as seen in Figure 5.25.



Figure 5.25: No Access Rights To Rename/Delete

But the students are granted write access only to certain communities where they can upload their assignments or home works. Here, the student Stefanie is given write access to the community named Assignments which is inside the Human Machine Interaction community. To the Human Machine Interaction community, she is added by Prof.Thomas who first makes her a friend and then adds her to the community. When Prof.Thoams sends friend request to Stefanie, she then receives a notification of the friendship request sent by him. She accepts it by selecting the *green tick* button as seen in Figure 5.26.

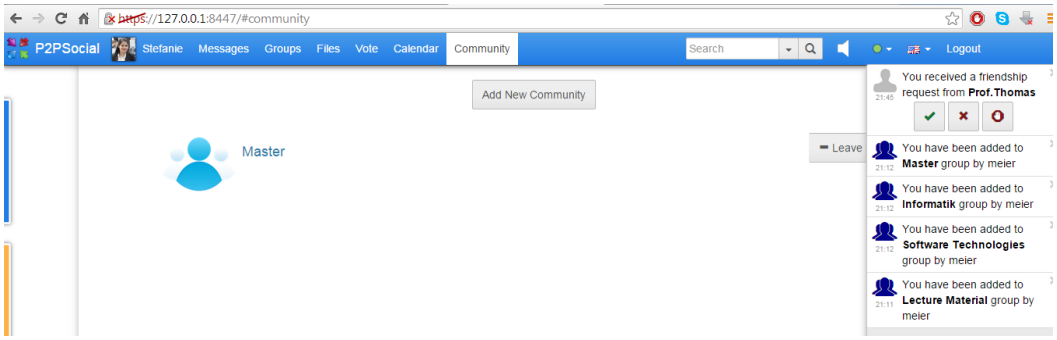


Figure 5.26: Friendship Request Notification

He then adds her to the community by clicking on **Add Friend** button which opens up a modal dialog showing his friends list as in Figure 5.27.

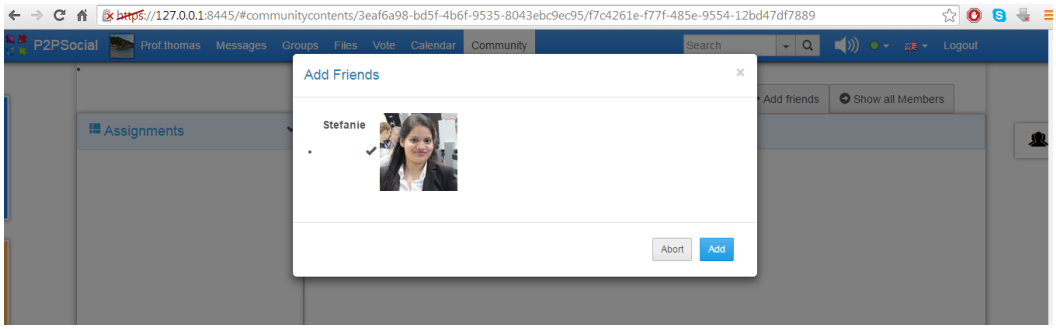


Figure 5.27: Add Friends Dialog

Assignments community which is an inner community of Human Machine Interaction community is for the students to upload their own home work assignments of that particular Human Machine Interaction course. Assignment files can be submitted by selecting the option Files from the drop down menu, which opens up a file selector dialog where one can browse for their files and upload them. The uploaded files can be deleted by clicking on the cross glyphicon symbol present to the right side of the file as shown in the Figure 5.28.

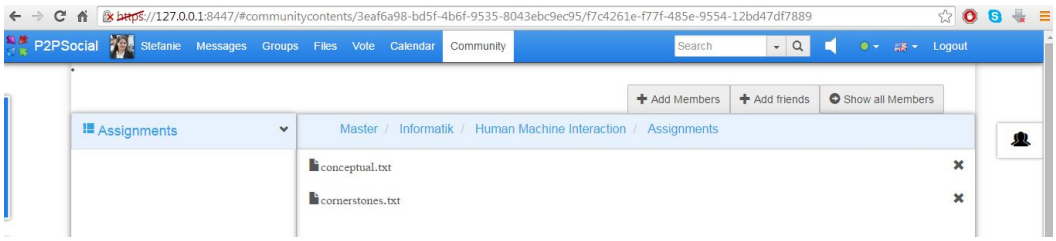


Figure 5.28: Files Upload

The uploaded files can be downloaded by just clicking on the file itself, which points to the url of the file where it is located and downloads it as shown in the Figure 5.29.

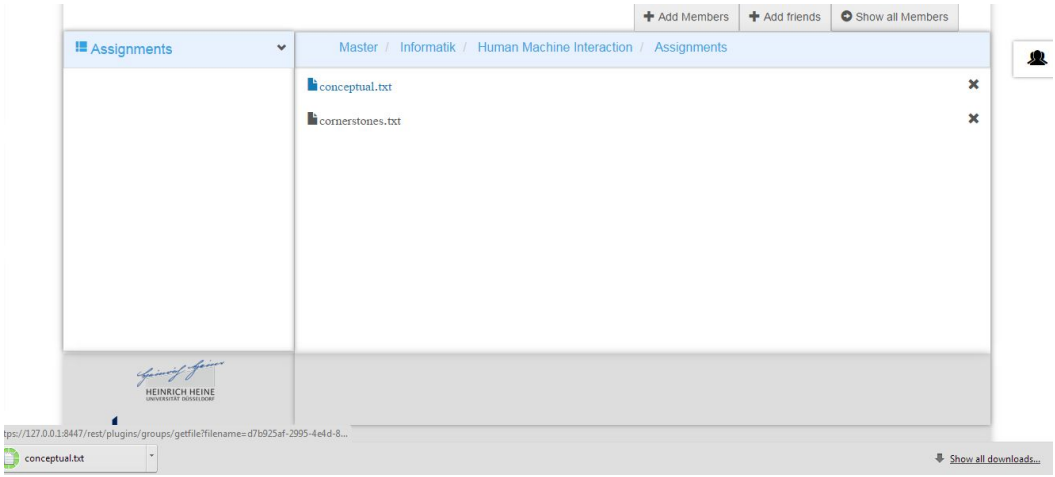


Figure 5.29: File Download

This section presented all the functionalities which have been tested and the screen shots that have been taken during the testing process. Testing helped to identify minor errors which have been corrected later.

6 Conclusion

This chapter summarizes the overall achievements in thesis and presents the future work to be done. This chapter is written by Christy Jovita Bibiana Arockiaraj.
In LifeSocial, there are various plugins available like messaging, chat, login, notification plugins.etc. In this thesis, the groups plugin is extended to add more features to it. The main goal of the theis is to create comunities so that user can join the community and share the documents and folders with other users in the same community. The main achievement of the thesis is creating inner level communities and implementing access control mechanisms. Some of the features added to the groups plugin are adding folder and files to any hierarchy level of the community. Moreover, some of the features that contributes the access control mechanism are renaming, deleting and creating communities, folder and files for a community. The user can invite other users to the community with read/write access. The user with write access can only delete or rename an item in the community. Eventhough there are some problems in this community plugins like deleting the community. The user cannot delete the inner community. The user has to leave the top level parent community to leave the community.
The future work is to add more features like sharing the calenders among the users in the community and adding forums. When deleting the community, an exception seems to occur when recovering all the related contents of the community. These are the features and issues that can be addressed in the future work.

Appendix

...

List of Figures

Bibliography

[1] OSGi Alliance. Osgi for developers. Technical report, 2015. Accessed online under `http://www.osgi.org/Main/OSGiForDevelopers`, fetched on 22. February 2015.

[2] Christy Jovita Bibiana Arockiaraj Kailaash Balachandran Navendu Barua Bing Gao Eshant Tulsidas Gour Christopher Holomek Thomas Jacobs Siddharth Kumar Bhagvanjibhai Kaneriya Maitri Modasia Weifeng Peng Chander Prakash Manasa Veena Rapolu Florian Schulte Rohan Sukumaran. Final report - a peer-to-peer framework for social networks, 2013.