LifeSocial - Masterpraktikum

Florian Gattung, Martin Kleine, Immanuel Pahlke

***Abstract*—This report describes the approach of the lab exercise project for LifeSocial in the winter term 2008/2009, which was organized by Multimedia Communications Lab at TU Darmstadt. It contains a detailed depiction about the two steps which were undertaken to cope with the main task. Furthermore, a short overview of activities beside the main task is presented.**

I. Introduction

A. LifeSocial

LifeSocial is a distributed platform for multimedia online communities based on P2P technology, developed at the Multimedia Communications Lab, TU Darmstadt. The challenge was to create a P2P based architecture which allows implementing the typical online community functions without the need for high-end infrastructure.

Based on FreePastry a generic, scalable, efficient and open-source P2P substrate, the aim for LifeSocial is to become part of the SkyNet Project "Towards a Sentient Distributed Network". SkyNet itself is a research project at Multimedia Communications Lab, TU Darmstadt and a prototype of a self-aware network that decides about steps to improve its existence without user interaction.

B. Participants

The team for this lab exercise consists of three students, Florian Gattung, Martin Kleine and Immanuel Pahlke, majoring Wirtschaftsinformatik (Diplom) at TU Darmstadt. The three students were advised by Kalman Graffi, research assistant at Multimedia Communications Lab, TU Darmstadt. Mr. Graffi, responsible for the assignment of tasks and the LifeSocial project in general, introduced the group into technical details and gave feedback on frequently arising questions. The participants were responsible for the the design and implementation of the plugins.

C. Assignment

Based on an existing prototype of LifeSocial, the assignment was primarily to plan, implement, test and document new plug-ins. Our principal object was to have something presentable for CeBIT exhibition 2009, held in March 2009. The target audience were business members on the one hand, to use LifeSocial for business communication, comparable to a groupware. On the other hand LifeSocial has to be geared towards the private business as well. People who spend their leisure time with online communication are attracted by multimedia enriched application content and functionality. LifeSocial in this case represents the underlying platform and therefore the focus of our project was to develop plug-ins for each target group.

II. Approach

The approach for the lab exercise consisted of several meetings with the adviser. The objective of the kick-off meeting on October 15th, 2008 was to introduces the group into the architecture of LifeSocial. Each of the main components and their collaboration was presented. The meeting was followed by several small group meetings, where the team presented single milestones to the adviser and discussed further issues. In this section, these two main phases will be presented in more detail.

A. Analysis of the Framework

To get a better and deeper understanding of LifeSocial with all its components, the first step was an analysis of the framework itself. Existing plug-ins were assayed and possible candidates for new plug-ins were carved out. Thereby the team kept the focus on business and private clients in mind. In the process of analysis we found out, that the most plug-in candidates have one characteristic in common. As members in a social network tend to communicate in groups, many plug-ins need the structure of group management and group messaging. Hence, the first plug-in to develop was the GroupEnabledPlugin. As that plug-in touched many facets of the LifeSocial platform, it was also suited to learn how to deal with each component in LifeSocial. The analysis phase ended with an agreement between the team and the advisor upon which plug-ins the team has to concentrate on.

B. Implementation

The implementation of the plug-ins always took place in the team. This enabled practicing the four-eye principle. Questions could be asked immediately and ideas as well as inspirations were exchanged directly. In order to use the implementation time effectively and efficiently, several coding sessions were accomplished over several days. Thereby important milestones could be finished within a session. In order to present our results, we will now describe each plug-in more detailed.

1) GroupEnabledPlugin: The GroupEnabledPlugin (see figure 1 in the appendix) is an abstract class which provides functionality that is commonly required for various kinds of plug-ins in a social network, especially for those which use communication and shared storage functionality among groups.

Overview

Plug-ins using the provided infrastructure can use the

methods joinGroup, leaveGroup and sendMessage to easily manage group behavior. Message exchange and storage is already provided in the background.

Number of groups per GroupEnabledPlugins
Every GroupEnabledPlugin is (by default) able to handle an arbitrary number of groups at the same time. Hence, most commands have the String groupName as the first parameter to identify the group a specific command should be applied for. This means that, for example in the GroupChatPlugin, a user can be in several chat rooms at the same time.

Relation of groups and plug-ins that implement GroupEnabledPlugins
By using the GroupEnabledPlugin all data is stored relative to the actually implementing plug-in. Assuming two GroupEnabledPlugins, e.g. the GroupChat- and the WhiteboardPlugin are instanciated by users and those join a group with the same name, e.g. "Session1". The messages send to the group "Session1" are always send as "messages for the group Session1 and plug-in GroupChatPlugin" and therefore do not interfere with each other.

Message Types
The main message type that is used to communicate between GroupEnabledPlugins is the GroupEnabledMessage. Usually, implementing plug-ins will extend this message type for their own specific requirements. Two GroupEnabledMessage extending classes are the GroupHistoryRequestMessage and the GroupHistoryResponseMessage, which are described in more detail in the following paragraph.

Message history
Message exchange is stored in a history which is also persisted to storage. When a new user joins a group, he reads the latest history from the storage and (automatically) sends a GroupHistoryRequestMessage to all users in the group. All peers receiving the message then respond with their latest version of the group history (GroupHistoryResponseMessage). The GroupHistoryResponseMessage is then used to fill the local version of the peer that just joined the group. The Message history is stored in the DHT using the SharedItem type GroupMessageHistoryItem. Even when a user goes offline for a while or joins a group later than other users, he will always have the same and full history of all messages that have been send to this group in the past.

Remembering the groups of a user
The group memberships of a specific user is persisted on shutting the network down. Therefore, a graceful shutdown using the quit command is very important. The groups of a specific user are stored in using the SharedItem type GroupNamesItem.

Remembering the users in a group
At a given time, all users that actually build the membership in a group are stored in the SharedItem of type GroupUsersItem. This item is always updated in the DHT when a user joins or leaves a group.

Sending a message
When a message is send to all members in the group, the GroupUsersItem is requested from the DHT, and on arrival, the message gets send. In order not to forget the message that is supposed to be send after the GroupUsersItem arrives (which is an asynchronous call), a special request type is required. This request type is implemented in the GroupMessageRequest class.

2) GroupChatPlugin: A plug-in that implements a group chat functionality. This allows different users to join a common group and to exchange text messages within this group.

The plug-in is realized as a GroupEnabledPlugin, using a specific GroupChatMessage type to exchange the text messages.

Available commands

`joinGroup`
This command is used to join a group in which a user would like to chat in.

`sendMessage`
This command can be used to send a message to a chat room.

`leaveGroup`
This command can be used to leave a chat room

`getAllMessages`
This command can be used to list all messages in a specified chat room.

Example command line code (use from several nodes):

```
groupenabledjoingroup GroupChatPlugin Chatroom
groupchatsendmessage Chatroom Hello EveryBody\n
 in ChatRoom, this is me!
groupchatsendmessage Chatroom Hello EveryBody,\n
 this is me again!
groupchatsendmessage Chatroom Hello EveryBody,\n
 this is me once more!!!!
groupchatgetallmessages Chatroom
```

3) WhiteBoardPlugin: A plug-in that implements the functionality of an shared whiteboard. This allows different users to join a common group and to exchange shape messages within this group. The messages are stored in the shared memory. Therefore it is possible that a user can join a group and get the actually status of the whiteboard session.

The Plug-in is realized as a GroupEnabledPlugin, using a specific WhiteboardMessage type to exchange the different shape messages in the group. A lot of functionality (group managemnt and group communication) is implemented in the GroupEnabledPlugin, and can be reused in the whiteboard plug-in. In the following we will describe the available commands and the implementation of the functionality.

Available commands

`joinGroup`

This command is used to join a group in which a user would like to share a whiteboard.

`sendWhitheboardMessages`

This command can be used to send a message to the shared whiteboard. The message contains the shape which should be added to the shared whiteboard. To describe the different shapes a shape-class-structure was implemented. This class structure is easily extendable for further shape types. Every shape type has its own console command to send a special shape message.

`leaveGroup`

This command can be used to leave a whiteboard session.

`getAllShapes`

This command can be used to get all shapes drawn on a special whiteboard.

Example command line code (use from several nodes):

```
groupenabledjoingroup WhiteboardPluginId Test
whiteboardsendpoint Test 128 2 1 5
whiteboardsendline Test 128 2 1 1 5 5
whiteboardsendpolyline Test 1 1 2 2 3 3 4 4 5 5
...
whiteboardgetallshapes Test
```

4) TicTacToePlugin: The good old game with a 3x3 playboard and X's and O's. Two players play against each other and viewers are able to watch the game whilst not able to actually participate. A console like textual representation of the playboard was implemented to display the actual game state.

It is implemented as a GroupEnabledPlugin. The first 2 users in the group are the players (X or 0), every user after them is only able to watch the game. To start a new game, simply join a new group with the TicTacToePlugin.

Table I and II show the structure of the playboard and an example configuration, respectively.

Table I
INDEX STRUCTURE OF THE PLAYBOARD

0	**1**	**2**
3	**4**	**5**
6	**7**	**8**

Table II
AN EXAMPLE CONFIGURATION OF THE PLAYBOARD

X	**0**	**X**
	X	**0**
		0

Available commands

`joinGroup`

A game is realized using the GroupEnabledPlugin architecture in the way that in one group, always one game is played. Hence, to start a new game, a new group has to be joined. The first two users in a group are automatically assigned as the players, whilst the rest of the group is only able to watch the game.

`leaveGroup`

This command can be used to stop playing a game

`isPlayer`

This command can be used to find out if the current user is a player in the given group

`isMyTurn`

This command can be used to find out if its ones own turn in this group

`getMySymbol`

Returns the own symbol (either "X" or "O") in a game. This command only works if oneself is a member in the group and oneself is a player in the group.

`getCurrentGameState`

This command is used to get the current game state (=what the playboard looks like). It works by parsing out all messages containing a move from the message history in the group and build up a game state object.

`makeMove`

Make a move on the current game. Sets the own symbol on the given field. After the move has been made successfully, the status of the game is checked (e.g. if somebody won the game).

`checkGameState`

Can be used to check the current status of the game. Returns -1 if the game is still running or 0/1 for the winning user (0="X" , 1="O").

Example command line code:

```
groupenabledjoingroup TicTacToePluginId Spiel
tictactoelistplayers Spiel
tictactoemakemove Spiel 2
```

5) VotePlugin: A plug-in that allows people to perform surveys or votes. has the functionality to set a specific question for a given vote, to add an unlimited number of

options to chose from and the possibility to set a vote mode to single or multi, which means that only one or several options can be voted for. Only the vote master (who is the person that starts the vote) can end the vote, but the vote can go on for an unlimited length even if the vote master is not online.

The VotePlugin is a GroupEnabledPlugin. It is using the group structure in the following way:

1) Everybody who wants to participate in a specific vote needs to join the same group
2) The first group user who send a VoteStartMessage to the group is the vote master. The VoteStartMessage contains all information about the vote itself (question, vote mode, options).
3) From an API perspective, a specific vote is always accessed by its groupName.

A vote consists of:

1) Question - the question the vote should answer
2) VoteMode - either "single" or "multi" (single meaning only one option may be selected, while multi means that several options can be selected.
3) Options - a set of options to choose from

Available commands

`joinGroup`

Join the group for the specified vote, either to start the vote (=being the vote master) or to participate in the vote.

Commands for the vote master

`setQuestion`

Sets the question for this vote. Works only if vote has not been started yet.

`setMode`

Sets the mode for this vote. Works only if vote has not been started yet. Mode can either be "single" or "multi".

`addOption`

Adds an option for this vote. Works only if vote has not been started yet

`delOption`

Deletes an option for this vote. Works only if vote has not been started yet and this option is already in the set of options

`startVote`

Starts the vote in the given group. Works only if a question was set, at least two options were added and a vote mode was set

`endVote`

End the voting in the given group. Can only be performed by the vote master.

Commands for the vote participants

`voteFor`

Choosing an option for the actual vote process. Marks a vote for a specified option. Works only if vote was already started and needs a valid option from the option set to be successful. If VoteMode = VoteMode.Single, only one vote will be valid. This command will fail if user has already voted for this option.

`getCurrentVoteStatus`

Getting the current vote status. VoteStatus.Before - voting has not started yet VoteStatus.Running - voting has started and not yet ended VoteStatus.After - voting has already ended.

`getVoteOptionSet`

Can be used to get the set of available options to choose from.

`isVoteMaster`

Can be used to find out if the current user is the vote master in a specified group.

`leaveGroup`

This command can be used to stop participating in a vote. Options that have been voted for are stored and will be used to generate the results once the vote is ended.

Example command line code:

```
groupenabledjoingroup VotePluginId Umfrage

votesetmode Umfrage single
votesetquestion Umfrage Wenn am nächsten\n
 Sonntag Bundestagswahl wäre, welche\n
 Partei würden Sie wählen?
voteaddoption Umfrage CDU
voteaddoption Umfrage SPD
voteaddoption Umfrage FDP
votestartvote Umfrage

votevotefor Umfrage CDU
votevotefor Umfrage SPD
voteendvote Umfrage
```

6) SchedulePlugin: The purpose of the SchedulePlugin is to enhance the LifeSocial community with the possibility to share and organize appointments. LifeSocial users can create, view and share appointments with each other. The SchedulePlugin builds on group messaging features provided by the GroupEnabledPlugin. The exchange of appointments is realized via serialized messages that are send similar to a group chat. Furthermore the specification and implementation of single appointments are intended and prepared to simplify an implementation of an import/export-interface via ical-format (for a detailed specification, see http://tools.ietf.org/html/rfc2445).

Available commands

`joinGroup`

This command can be used to enter a group of members sharing one calendar.

`leaveGroup`

Contrarily, the user can leave the group and stop sharing appointments with other members of this group

`listSchedules`

Lists all relevant appointments to the user. Relevant appointments are group appointments for groups, in which the user is a member. If the groupname is passed as parameter, only appointments relevant for this group are displayed.

`createSchedule`

With this command a schedule is created. All relevant data (title, group, place, start, end, repetitions) must be specified. Optionally the user may pass a description for the particular appointment. The created appointment will further be published to other group members.

`changeSchedule`

This command allows every user of the group to change details of a particular appointment. All relevant data (title, group, place, start, end, repetitions) must be specified. Optionally the user may pass a description for the particular appointment. The changed data will further be published to other group members.

`deleteSchedule`

Individual appointments can be deleted by every user in the group. The deleted appointment will not be deleted immediately, but marked with a flag to signal the deletion to other group members.

Example command line code (usage from several nodes is possible):

```
scheduleplugincreateschedule Family Birthday\n
 Berlin 12.12.1970 00:00:00 12.12.1970 23:59:59\n
 4 Today is Emily's birthday.
scheduleplugin listschedules Family
scheduleplugin changechedule 123456789 Family\n
 Birthday Berlin 12.12.1972 00:00:00 12.12.1972\n
 23:59:59 4 Emily is not that old.
scheduleplugin listschedules
scheduleplugin deletechedule 123456789 Family
```

7) AddressBookPlugin: The AddressBookPlugin enables a user to build up his own address book based on LifeSocial members as well as non-members. He has the possibility to enrich his own address book by meta data, another LifeSocial user has not provided. Furthermore an import and export of the user's address book is possible via vcard-format (see specification on http://www.ietf.org/html.charters/vcarddav-charter.html).

Available commands

`updateContact`

This command can be used to update the user's address book with actual data, the LifeSocial user has published.

`addProperty`

With adding a specific property to a contact, the user can amend his address book with additional meta data not provided by his contact.

`importContact`

To simplify the filling and updating procedure of a user's address book, he can use the import function to import data in vcard-format.

`exportContacts`

Otherwise it is also possible to export the enriched user's address book in vcf-files to easily transfer them into another personal information manager, e.g. Outlook.

`getMetaData`

Once specified in the address book, the user has the possibility to list all data provided for a given user.

Example command line code (usage from several nodes is possible):

```
addressbookupdatecontact Joe
addressbookaddproperty Joe wife Mary
addressbookexportcontacts c:\contacts\
addressbookgetmetadata Joe wife
addressbookgetmetadata Joe
addressbookimportcontact c:\contacts\joe.vcf
```

8) FileTransferPlugin: The File-Transfer plug-in can be used for sending files to the users of the LifeSocial network. Because this plug-in is not an GroupEnabledPlugin we will document the implementation in more detail. Therefore we will first describe what functions the File-Transfer plug-in should provide. Then we describe how this functionality is implemented. Finally, the API and the corresponding commands for the user-console are defined.

The requirements can be described as follows:

- Create a list of shared files for every user.
- Every user should be allowed to append and remove filepaths to the own list of shared files.
- A user should be able to request the list of shared files from every user.
- The user should be able to exchange files with other users.
- The requested and finally received files should be stored.
- Undelivered messages should be stored in the network (DHT).

The plug-in should use both the shared network storage

to store the lists of shared files for every user and direct communication with messaging and local message administration to request the files from a given user and the final file-transfer.

First we identify different types of objects we have to deal with:

1) A list of filepaths which is associated with a user an contains the files which are enabled by the user for the file-transfer
2) File messages which contains the data and the name of the special file

For holding a list of shared files for a user we implement a new class by extending the class SharedItem. The file message is implemented by extension of a superclass for all messages (BasicMessage). The payload of the new messagetype contains a Byte-Array which stores the data of the specified file. The header was extended by the username to whom the file has to be send and the filename. Furthermore a new class named FileTransferPlugin is created. This class implements the the common interface for all LifeSocial plug-ins. The implementation of the required functionality of the plug-in is described in the following:

Add a filepath to the filepath-list
To add a filepath to the list of enabled shared files, the list first has to be created. In order to create a list of filepaths, a storage key with the name of the user is created and the object associated with this storage key is requested. The StorageDispatcher indicates whether the data object exists or not. If no object with this key exists, a new instance of a SharedFilepathList-Item is created. The new filepath is inserted into the payload of the object and stored using the Storage Dispatcher.

Remove a filepath from the Filepath-List
In order to remove a filepath, the storage key of the users FilepathList-Item is build using the username. Then the SharedItemFilepathList data object associated with this storage key is retrieved. The special filepath is removed out of the payload of the object and the SharedFilepathList is stored back in the network using the Storage Dispatcher.

Request the Filepath-List from a user
The storage key of the users FilepathList-Item is build using the username. Then the SharedItemFilepathList data object associated with this storage key is retrieved.

Send/receive file-transfer messages
In order to send a file-transfer message to a user of the LifeSocial network, we use the Message Dispatcher for direct Message-Communication. The file requests and file transfer messages sent by other users, arrive in the Information Cache. A class named PluginMsgsCacheChecker checks for new plug-in messages for a certain plug-in and then calls a method to notify that new messages are available.

Undelivered messages should be stored
Message Dispatcher provided the method where given a plug-in identifier the undelivered plug-in messages are loaded into the Information Cache.

III. Further Activities

The core assignment exceeding, the team also included some side work. This side work consists of fixing detected bugs in the framework as well as extending LifeSocial

A. Bugfixing & Extension of the Framework

The version of LifeSocial that we were given had recently been extended with Security. The Security appeard to be working fine, but some other new problems arised in the messaging system which made LifeSocial hardly usable at all. After intensive debugging, the problem was solved.

B. Dummy for Automated Creation of a Testing Environment

The typical start-up process of LifeSocial would usually take some time to simply enter some common commands. The newly implemented command createenvironment takes care of createing some users, have them join some groups and adding each other as friends.

C. Documentation

Not only our own work was well documented, partially using UML, but we also created a screencast video tutorial on how to initially set up LifeSocial when working with Eclipse on the KOM subversion server. The Screencast can be requested from Kalman Gaffi.

IV. Conclusion

The work of the team can be wrapped up with the following points. First, a bunch of new plug-ins for LifeSocial was designed and implemented. One of them, the GroupEnabledPlugin delivers basic functionality for further development and a structure which is necessary for many further plug-ins in the social community context. Besides a plugin with very founding functionality, several plug-ins focusing business as well as private audience were realized. Together with these plug-ins, common interfaces for communication with an overlaying GUI framework were included.

Additionally to the basic work, a component for the automated creation of a test environment was designed. Finally, a more or less severe bug concerning the handling of node was fixed.

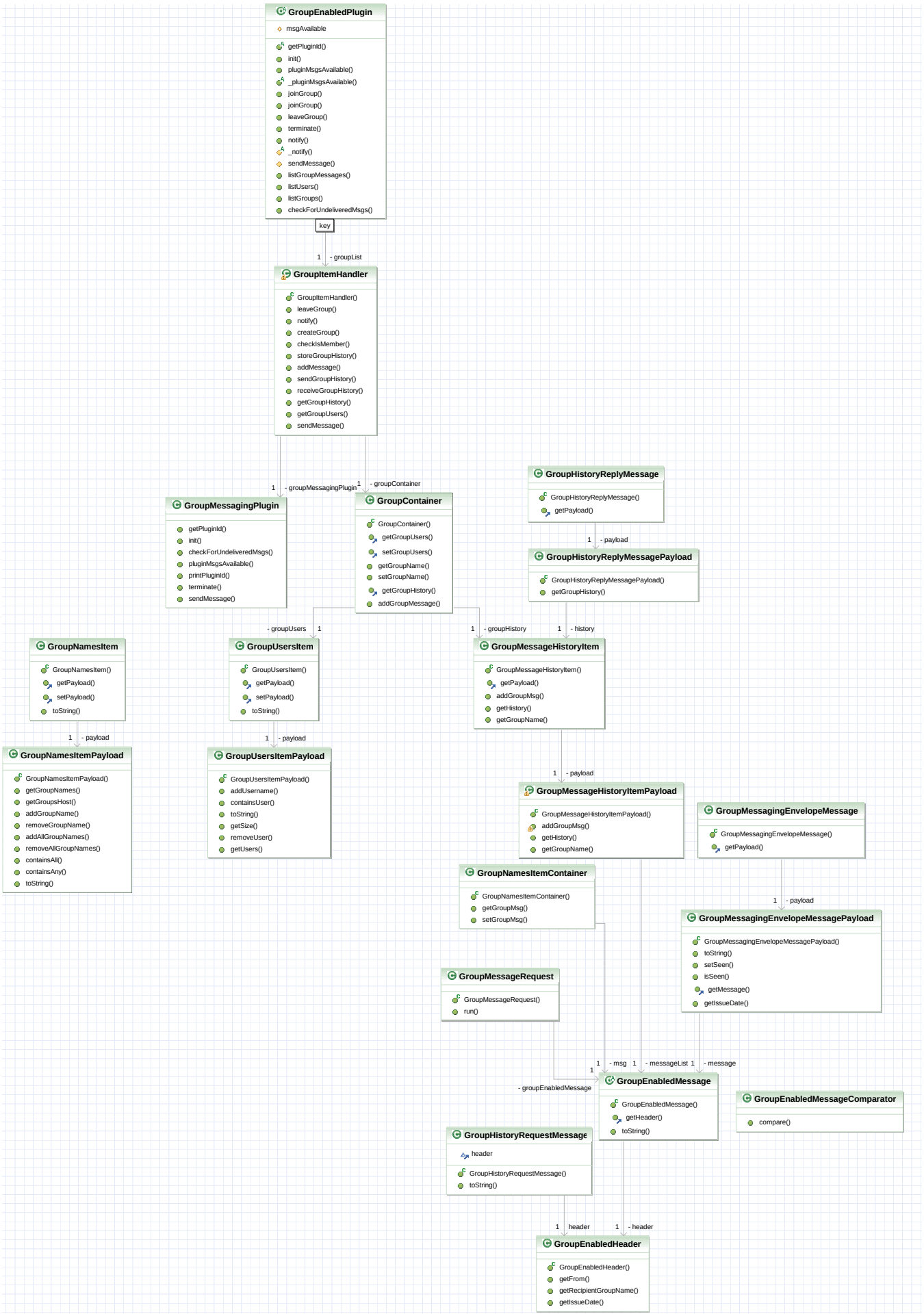


Figure 1. GroupEnabeldPlugin UML Diagram