

Technische Universität Darmstadt

Department of Electrical Engineering and Information Technology
Department of Computer Science (Adjunct Professor)
Multimedia Communications Lab
Prof. Dr.-Ing. Ralf Steinmetz

A Peer-to-Peer based Framework for Large-Scale Social Networking Platforms

Diploma Thesis

Submitted by

Sergey Podrajanski

on
15. June 2008

Advisor: Prof. Dr.-Ing. Ralf Steinmetz

Tutor: Kálmán György Graffi

KOM-D-0316

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 15 Juni 2008 . . . . . .

Contents

Ehrenwörtliche Erklärung **iii**

1 Introduction **1**
- 1.1 Introduction . . . 1
- 1.2 Online Community . . . 2
- 1.3 Motivation . . . 3
- 1.4 History . . . 4
- 1.5 The Peer-to-Peer Paradigm . . . 5
- 1.6 Related Work . . . 9
- 1.7 Goals of the Thesis . . . 9

2 Functionality Analysis **11**
- 2.1 Target Groups of the Platforms . . . 11
  - 2.1.1 StudiVZ . . . 11
  - 2.1.2 MySpace . . . 12
  - 2.1.3 Facebook . . . 12
  - 2.1.4 Xing . . . 14
- 2.2 Common Functionality of the Platforms . . . 14
  - 2.2.1 Registration . . . 14
  - 2.2.2 Profile . . . 15
  - 2.2.3 Friends/Contact List . . . 17
  - 2.2.4 Photos . . . 18
  - 2.2.5 Messaging . . . 19
  - 2.2.6 Groups . . . 20
  - 2.2.7 Networks . . . 23
  - 2.2.8 Search . . . 23
  - 2.2.9 Settings . . . 26
- 2.3 Specific Functionality of MySpace . . . 27

2.4 Specific Functionality of Facebook 28
2.5 Specific Functionality of StudiVZ 29
2.6 Specific Functionality of Xing 30
2.7 New Useful Functions 31
2.8 Summary of Functional Analysis 33
2.8.1 Mandatory Functions 33
2.8.2 Optional Functions 34
2.8.3 Nice-to-have-Functions 35
2.9 Summary 36

3 Peer-to-Peer based Prototype Requirements 37
3.1 Functional Requirements 37
3.2 Technical Requirements 44

4 Architecture Design 47
4.1 Architecture Overview 47
4.1.1 General Idea 47
4.1.2 Peer-to-Peer based architecture 49
4.2 Data Structures 60
4.2.1 Plugins Communication 60
4.2.2 Distributed Linked Lists 63
4.2.3 Example Scenario of the Architecture 66

5 Prototype Design and Implementation 71
5.1 Architecture Components 71
5.1.1 Pastry 74
5.1.2 FreePastry 75
5.1.3 PAST 75
5.1.4 Storage Dispatcher and Message Dispatcher 78
5.1.5 Information Cache 79
5.1.6 Assumptions 81
5.1.7 Security issues 82
5.2 Online Community Prototype Design 82
5.3 LiveSocial.KOM Plugins 84
5.3.1 Login Plugin 84
5.3.2 Profile Plugin 87
5.3.3 Search Plugin 88
5.3.4 Messaging Plugin 91

5.3.5 Friends Plugin 92
5.3.6 Live Chat Plugin 95
5.3.7 Photos Plugin 96
5.3.8 Groups Plugin 99
5.4 User Interface 104

6 Architecture Evaluation **107**
6.1 Robustness 107
6.2 Storage Behavior 108
6.3 Scalability 109
6.4 Extendability 109
6.5 Event Propagation 112

7 Conclusions **117**
7.1 Conclusions 117
7.2 Future Work 119

List of Figures

1.1 Traffic Rank 5

2.1 StudiVZ Screenshot 12
2.2 MySpace Screenshot 13
2.3 Facebook Screenshot 13
2.4 Xing Screenshot 14

4.1 Service Layers Model 48
4.2 Peer-to-Peer Based Architecture 50
4.3 Plugin to Plugin Communication 62
4.4 SharedItem and Plugin Data Object 63
4.5 Distributed Linked List Example 65
4.6 Layers Communication Example 69

5.1 Applying Peer-to-Peer Architecture 72
5.2 Plugins Layer 73
5.3 LoginItem Structure 87
5.4 UserProfile Structure 88
5.5 SharedItemUserList Structure 89
5.6 Photo Plugin Data Structure 97
5.7 GroupNamesHolder Structure 100
5.8 GroupMsg Structure 101

6.1 Facebook Error 108

Chapter 1

Introduction

1.1 Introduction

Online communities like MySpace [MyS06], Facebook [Fac04], StudiVZ [Stu05], Xing[Xin03] and some others became highly popular in the last few years. Usually online communities are web-based applications which provide different services for the members of a certain community. These services help to establish and to support social networking between people. Sometimes online communities are also called online social networks.

Every Internet user may become a member of any of the existing online communities. The main function of the online social networks is enabling the community members to interact in any way they would like to. Traditionally, this goal can be reached by providing various web-based services like: messaging, chat, sharing of personal information and images, filesharing, blogging, discussion groups, online groups of interest, search for new users using certain criteria etc. Currently popular online communities are usually client/server based web applications.

Since the first such client/server platform was made available, the number of users of these online communities has been continually growing. For some of these communities the number of members reached tens of millions. This development has a considerable impact on the amount of data which must be stored on the servers and administration complexity of the systems. The costs of server administration are growing with the increasing number of members of these communities since the data traffic and storage demand are growing proportional to the number of platform's users.

Unlike the client/server platform architecture, a Peer-to-Peer based architecture for an online community does not involve central storage and administration entity. All network participants, which are also called peers, in a Peer-to-Peer network are equivalent. Each peer operates as a server and as a client at the same time. This involves making

local resources like memory, disk space or bandwidth available for usage by other peers. Peer-to-Peer technology offers a number of considerable advantages in comparison to client/server technology in terms of administration cost, scalability and robustness. Since there is no central entity in a peer-to-peer network almost no administration costs arise. All system data is distributed among the peers. Besides, Peer-to-Peer based applications have a better scalability than traditional client/server platforms.

Considering currently high administration costs of the popular client/server based online communities, a Peer-to-Peer based architecture appears to be beneficial and promising for this type of web-based applications.

In scope of this diploma thesis the feasibility of using the Peer-to-Peer technology for online communities platforms is researched. This involes following steps: First the functions and services of four currently popular client/server based online communities are examined. Afterwards the minimum requirements for a Peer-to-Peer based online community are defined. As the next step an appropriate Peer-to-Peer based architecture is built and evaluated. Prototype implementation of a Peer-to-Peer based online community using the results of the previous parts completes the work.

1.2 Online Community

According to [Cor05], a community is a group of people, which have something in common. The members of this group feel connected to each other over these common things, which can be interests, hobbies or anything else. Another definition is given in [SXY07]: "Online community or virtual community is a group of people communicating or interacting with each other by means of information technologies, typically the Internet, rather than in person". Community members can communicate with each other during meetings or over communication facilities like telephone, letters etc. A good example for a traditional community is a chess club where members of the club arrange periodical meetings, where they play chess, communicate and exchange information. An online community, as it is described in [Kim00], does not differ much from a real community. Both online community and a real community involve building relations between people who has something considerable in common. The author of [Kim00] states that an online community is a community that exists online. Members of an online community share a common interest or belong to a group of people, which can be identified using certain criteria such as hobby, political or religious views, life situation, professional activity or city of living. The Internet modifies the way community members can interact with each other. The web technology offers an easy way to communicate and to exchange various information. Web platforms provide the necessary services to make users be able to explore the interests and

activities of other community members and to exchange various data. Members of an online community can interact with a subset of community members and control the access to their personal information [Kim00].

We define an online community as a group of people which share a common computer based and Internet based system to interact and exchange any art of information. Every member of the community can belong to a subset of community members which share a common interest or belong to a group which can be identified using certain criteria.

1.3 Motivation

According to [Cha07] annual administration expenses for the Facebook vendor are estimated to be $\$1.05$ per single platform user. The total number of registered online community members for Facebook platform was estimated to be 60 Million in 2007 [Sto07]. It is straight forward to assume that administration costs which Facebook vendor has to bear sum to about 60 Million of dollars per year. The most of these expenses are due to characteristics of a client/server architecture. More precisely, these are the server administration costs. To decrease or completely avoid high server administration costs, an alternative technology has to be used as an architecture basis.

Peer-to-Peer architecture for an online community platform appears to require less administration effort than a client/server based architecture. Since there is no central entity in a Peer-to-Peer network, all nodes (peers) are equivalent. The administration costs are uniformly distributed among all peers. The platform vendor does not have to provide storage capacities since all system data is also distributed among single peers. The properties of a Peer-to-Peer network promise to enable the implementation of all necessary services which are currently offered by online communities.

Online social networks can connect people at low cost. The usage of the most online platforms is free. This characteristic of online social networking can be interesting for small companies which would like to expand their contact database. Businesses can also benefit from using online social networks for advertising and finding new business partners and customers all over the world.

The benefits of social networking based on computer networks are obviously very considerable and promise to grow in the future. People, communities and businesses can benefit from using or being involved into social networking. The communication over social networking platforms like StudiVZ is free of charge for the platform's users but becoming expensive for the vendor of the platform. Due to client/server architecture, the administration costs of the system can be very high. A Peer-to-Peer based architecture for online social networking platforms is a vision which is aimed to keep the admin-

istration and maintenance costs of such systems constantly low and the systems easily scalable. The system's performance thereby is expected to be at least that high as with a client/server based architecture. These new characteristics could place a new Peer-to-Peer based system in a beneficial position on the market in comparison to client/server based social networking platforms.

1.4 History

The idea of using personal computers linked into a computer network as a basis for social networking was suggested already in 1978 by Roxanne Hiltz and Murray Turoff. The first social networking websites appeared in 1990's (e.g. Classmates.com(1995) [Con95]) created by Randy Conrads, which provided the possibility for members to find their friends from high school, college, work or army and to keep in touch with them. After those first attempts of creating social networking websites, a considerable number of different web-based applications were made available on the Internet or on local computer networks. Such platforms were more or less popular usually within a certain country.

There are now a considerable number of web-based platforms for social networking available for all Internet users. The most of them were originally designed as non-commercial applications and charge no fees for using the systems or for being members of community. This characteristic of early online social networking platforms made the vendors of new commercial platforms not charge any fees as well, in order to remain in the competition.

The most popular ones are MySpace [MyS06] and Facebook [Fac04] in English speaking countries and additionally StudiVZ [Stu05] in German speaking countries. MySpace and Facebook are considered the biggest competitors on the market. The MySpace online platform was founded in August 2003 and offered different social networking services to all Internet users. MySpace website was designed and created by the Internet company eUniverse. The very first users of MySpace were employees of the company eUniverse. After making the system available on the Internet the number of users began to grow rapidly. In 2007 the number of registered members of MySpace reached 100,000,000 [Cas06]. According to Alexa.com [Com96], MySpace is the sixth world's popular website in English-language countries and the third most popular website in the United States. The biggest competitor of MySpace, Facebook, was first launched on February 4th 2004. The creator of Facebook was a Harvard student Mark Zuckerberg, who originally created the site for Harvard University network use only. After some time Facebook was expanded to other schools and universities in the United States. The main users of Facebook at the beginning were college and university students. In 2007 the

number of registered users had reached more than 60,000,000 [Sto07] and is growing permanently. According to Alexa.com [Com96], MySpace and Facebook are the most popular hosts in the web (state on 10th May 2008).

The Figure 1.1 shows the traffic ranking of MySpace, Facebook, StudiVZ and Xing on the basis of the pages daily traffic. One can see on the graph that MySpace and Facebook are placed in the top ten websites world wide in respect to the website daily traffic [Com96].

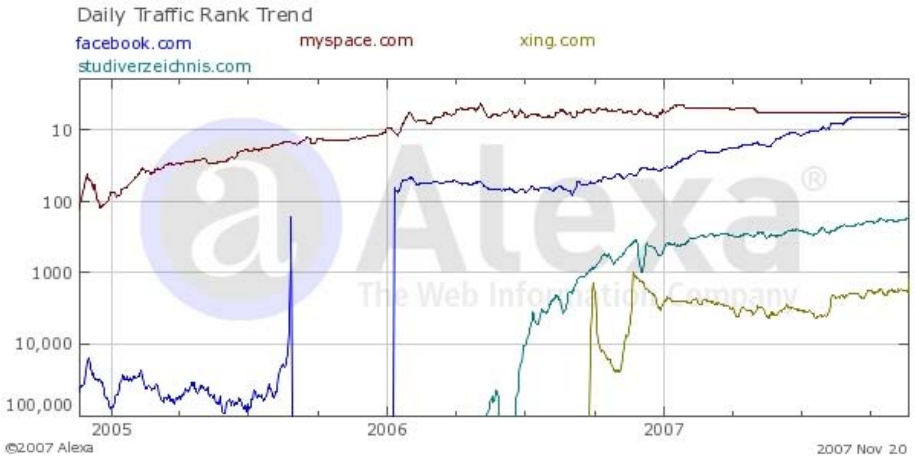


Figure 1.1: Traffic Rank

1.5 The Peer-to-Peer Paradigm

Originally designed for file exchanging applications, Peer-to-Peer technology was increasingly used in other types of Internet-based applications. A good example of a successful use of Peer-to-Peer technology is Skype. Based on Peer-to-Peer technology Skype offers three general services: VoIP enables two Skype users to build a voice-based audio connection which is used like an common telephone connection, instant messaging allows to exchange text-based messages between two or more users, a direct file transfer enables users to transfer files from user to user if both sites are agreed with it [AH06]. By some Internet providers estimated, about 50% of the entire Internet traffic is due to Peer-to-Peer applications. According to [SW05], a Peer-to-Peer system is defined as a high distributed, decentralized and self-organized system of (mostly) equal entities (peers). An alternative definition of a Peer-to-Peer system is given in [ATS04]: "Peer-to-Peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and

bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance without requiring the intermediation or support of a global centralized server or authority".

The properties for a typical Peer-to-Peer system can be summarized as follows [SW05]:

1. self-organisation
2. resource sharing
3. equality of all peers

A Peer-to-Peer system is usually implemented by building a Peer-to-Peer overlay network in the Internet. A Peer-to-Peer overlay network can be defined as an algorithm for communication between peers. An overlay network is a virtual network containing connections between peers and based on an underlay network. The underlay network provides the necessary network infrastructure and communication protocols [SW05]. Usually the Internet is used as an underlay network.

There is a number of criteria which can be used to distinguish different types of Peer-to-Peer networks. The structuring of a network is one possible criteria. Peer-to-Peer networks can be classified in two different groups:

1. structured Peer-to-Peer networks
2. unstructured Peer-to-Peer networks

Unstructured Peer-to-Peer networks are denoted by absence of any structure in the network. In the first generation of Peer-to-Peer based applications a central server entity was used to execute lookups of data locations. After executing these lookups, a direct communication between peers was established and the data was exchanged [SW05]. The major disadvantage of these approach was that there was still a single point of failure in the Peer-to-Peer network. If the central lookup server was unavailable, no communication between single peers could take place. A better approach in terms of network decentralization was used in Gnutella [Gnu01]. Peers could retrieve information about the distributed resources by using flooding techniques. A peer (requester) sends a request to all directly connected peers. The peers send back a data package which contains information about the resources provided by these peers. Every peer forwards the request to all directly connected peers except the requester. In general, queries were sent to all network participants until the search resource was located [LPC+05]. Unlike the first approach which is using a central server entity this approach was not affected by the single

point of failure problem. However, there was an enormous bandwidth consumption in consequence of flooding techniques used for searching resources.

In contrast to unstructured Peer-to-Peer networks, structured Peer-to-Peer networks have a certain topology. The peers in a structured Peer-to-Peer network hold information about a set of peers, which are usually the direct neighbors of this peer. Structured Peer-to-Peer networks are completely decentralized and self-organized [LPC+05]. As a solution of the resource searching problem a distributed indexing approach is used. This approach is called Distributed Hash Tables (DHT). In [Sar07] DHT is defined as follows:"DHT is a distributed data structure which holds key and value pairs in a completely distributed manner. DHT puts each key and value pair on a single node only". Originally DHTs were developed to provide distributed indexing of the resources but also increased the scalability of the Peer-to-Peer networks [SW05]. The concept of distributed hash tables uses the advantages of efficient storage and look up functionality provided by hash tables. The difference to common hash tables is that the entire hash table is divided into buckets and every bucket is stored on a certain peer. Each peer in a structured Peer-to-Peer network is responsible for a bucket of a hash table. DHT tables help to clearly assign responsibilities to single peers (nodes). This makes it easy to find data stored in a structured Peer-to-Peer network since there is always a peer or a set of peers which are responsible for this data. The assignment of the responsibilities is based on peers' identifiers. Every peer has a unique identifier. The uniqueness of the identifier can be achieved by using the IP-address of the peer as a basis for calculation of the identifier. The routing in such networks is more efficient than in unstructured Peer-to-Peer networks since every peer administrates a part of routing table which is permanently updated while some peers join or leave the network.

Peer-to-Peer technology appears to be suitable for implementing of an online community platform. As stated in [SW05] the most of the future Internet-based applications should meet following three requirements:

- scalability
- flexibility
- security and reliability

Scalability in structured Peer-to-Peer overlay networks is given by the complete decentralization of the network. Each new peer contributes resources, which can be used by other peers for storage of data. There is no central entity which has to provide e.g. bandwidth, memory or calculating capacity. The fact that every new peer not only utilizes but

also contributes resources makes Peer-to-Peer based online platform highly scalable. The flexibility of an Internet-based application is given by multiple properties:

- extendability
- substitutability

Extendability is given if an application can be easily extended with new modules which provide new services. Substitutability is denoted by the degree of easiness with which the most of the application components can be substituted by other may be better qualified modules. Flexibility of the system depends mostly on the quality of the architecture which forms the basis of the application.

Security and reliability of a Peer-to-Peer based application is given by the quality of the Peer-to-Peer overlay network as well as by the quality of the application implementation build on top of the network. A short overview over the security issues in a Peer-to-Peer systems is given in [Wal03].

Peer-to-Peer technology qualifies itself as a basis for an implementation of an online community because it provides the necessary scalability and flexibility. A structured Peer-to-Peer overlay network can be dynamically changed. New peers can join the network any time and the active peers can leave the network without having an impact on the network performance. Data in a Peer-to-Peer network is stored distributed. Generally, Peer-to-Peer technology provides the two core operations which are necessary for implementing of the most typical online community services: storage and communication. The storage capacity is provided by using algorithms for distributed data storage based on DHT. Communication capacity is given by direct exchanging of messages between peers. To make sure that the data stored distributed in a Peer-to-Peer network is permanently available, we have to ensure that a certain number of peers are permanently in the network. In an online community with many millions of users a considerable number of users are always online. This property ensures that the data stored and replicated over the active peers will permanently be available on the network. Some dummy peers can be joined to the network to reduce the possibility of data lost. An appropriate architecture which is built on top of a Peer-to-Peer overlay network should provide the necessary flexibility, reliability and security on the application level. The low administration costs concludes the list of arguments which qualifies Peer-to-Peer technology as an appropriate technology for building architecture for an online community platform.

1.6 Related Work

Peer-to-Peer technology is widely used and is a basis for various applications. Each of the existing Peer-to-Peer based systems focuses on providing a certain service. For example Skype [AH06] provides VoIP based telephony service and messaging service. A number of Peer-to-Peer applications like Gnutella [Gnu01] enable users to share data. The main purpose of such Peer-to-Peer applications is filesharing. The Peer-to-Peer video streaming system, Zattoo [Zat06], enables users to watch european television channels online. Some Peer-to-Peer applications provide storage service for the users. A good example of such an application is Wuala [Wua04] which provides an "online hard disk" for its users. The requirements for every of the listed applications are various. A Peer-to-Peer based online community has not been implemented on basis of Peer-to-Peer technology before. The functions and services offered by an online community are very different. The Peer-to-Peer architecture designed in scope of this thesis allows to combine various services in a single Peer-to-Peer application. The requirements for such a system differ from requirements of the Peer-to-Peer systems which focus on providing only a certain type of service. Unlike the existing Peer-to-Peer applications, a Peer-to-Peer architecture developed in scope of this thesis can be extended with different services including filesharing, VoIP and video streaming functionality. The design of a Peer-to-Peer based online community poses a row of challenges. An universal Peer-to-Peer based architecture with storage and communication service is required in order to implement an extendable Peer-to-Peer based online community platform.

1.7 Goals of the Thesis

The goals which are achieved in scope of this diploma thesis were defined as follows:

1. to analyze the functionality of four currently popular online communities Facebook, MySpace, StudiVZ and Xing
2. to determine the functional and technical requirements for a new Peer-to-Peer based online community
3. to develop a Peer-to-Peer based architecture which allows implementing of the typical online community functions
4. to provide a high extendibility of the system at the architecture level
5. to implement an online community platform prototype with basic functionality

This work is divided into chapters according to the given list of goals.

Peer-to-Peer technology can bring advantages for the platform's vendor in terms of radical reduction of system's administration cost. The successful achievement of the goals listed above qualifies Peer-to-Peer based online community platform as a competitive alternative to the client/server technology.

Chapter 2

Functionality Analysis

In this chapter an analysis of the functions provided by four currently popular client/server based online community platforms will be presented. First, the main functionality of MySpace, Facebook, StudiVZ and Xing will be captured. In the first section the target groups and the common functionality of the platforms will be described. Afterwards, the platform specific functions will be listed. Then some new useful functions and features which can be implemented in scope of a social networking platform will be suggested. In the last part of this chapter all listed functions will be classified into different categories considering the popularity and appearance of the functions in all four platforms.

2.1 Target Groups of the Platforms

Online communities are usually designed for a certain group of people denoted by common characteristics. Some of the online communities were designed for students, others were created for businessmen. In this section the target groups of the four examined platforms are described.

2.1.1 StudiVZ

StudiVZ is the most popular online community in German speaking countries. It was originally designed for college and high school students from Germany, Austria or Switzerland [Stu05]. The original name of the platform is "Studiverzeichnis". In 2007 StudiVZ was bought by Holtzbrinck-Group having more than 1 Million registered users [Stö07]. The most of them come from Germany, Austria and Switzerland. Figure 2.1 demonstrates a StudiVZ screenshot.

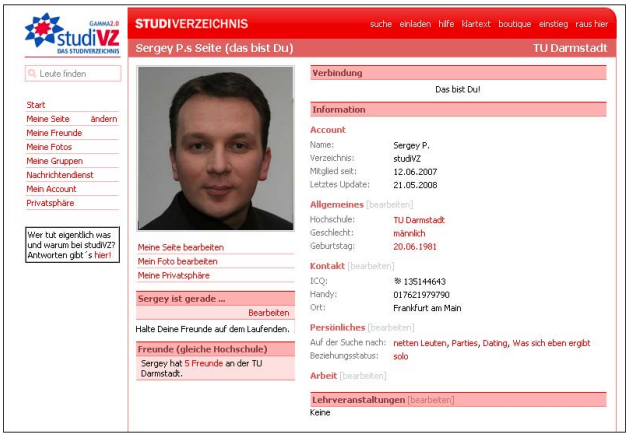


Figure 2.1: StudiVZ Screenshot

2.1.2 MySpace

MySpace is currently the biggest online community in respect to the number of users. MySpace was from the beginning a commercial project and was designed to enable the members to interact with any other members and to invite other people to become a member of MySpace community. MySpace can be considered as an appropriate system for:

- all friends, who would like to communicate online
- all singles, who would like to meet other singles
- people, who would like to keep in touch with their families and relatives
- all businessmen and colleague, who would like to build networks
- all who would like to find his/her schoolmates or old friends and keep in touch with them

The one of the biggest US media companies "News Corporation" acquired "Intermix Media", owner of Myspace, in 2005 for 580 Millions US dollars [New05]. Some analysts believe that the market value of MySpace could reach the point between 10 and 20 billions US dollars within a few years [Sil06].

2.1.3 Facebook

Facebook is an open platform available for every Internet user. Originally the online platform was designed for students of Harvard university. One of the students of Harvard

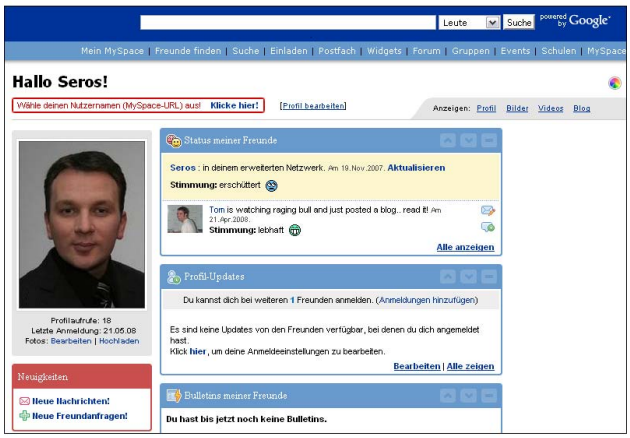


Figure 2.2: MySpace Screenshot

university named Mark Zuckerberg created the first version of the platform and invited his student fellows to join the community. Some time later students of other universities and colleges joined the community. The process of gaining new members moves on until today. Now Facebook is open for every Internet user who is willing to join the community. Below you can see a figure that shows a screen shot of the platform's web page.

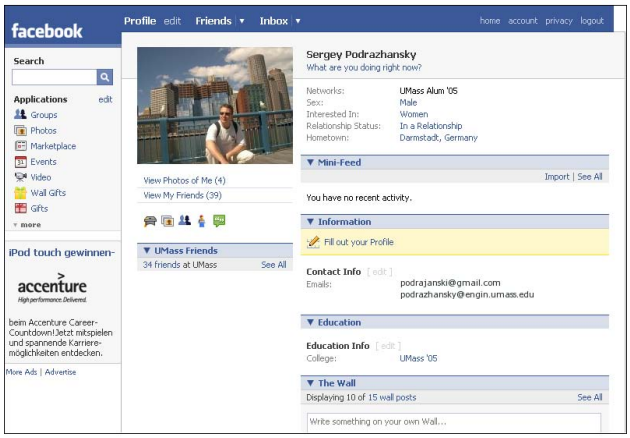


Figure 2.3: Facebook Screenshot

2.1.4 Xing

Xing is designed as job-related social network for business contacts. This is a big difference to other platforms like StudiVZ and Facebook, which were designed to enable college students to interact and communicate. The main goal of Xing is to enable businessmen to find other businessmen, specialist and tradesmen in any possible branch and to communicate with each other. The head office of Xing is located in Hamburg, Germany. Xing is also represented in Barcelona, Spain and Beijing, China.



Figure 2.4: Xing Screenshot

2.2 Common Functionality of the Platforms

Online communities provide various possibilities for communication and information exchange for their members. Some of these functions are offered by all examined online communities. In this section all such common functions are captured.

2.2.1 Registration

The use of all four platforms is free of charge and available for every Internet user. Following personal information is mostly required for registration:

1. first name, last name, gender, date of birth
2. email address(es), country, state, ZIP code
3. the email address of the user is usually used as login name

4. the user has to create a personal secret password to be able to access his/her account
5. after registration the user has to confirm the ownership of his/her email account by using a hyper link sent in an email message
6. to ensure the physical presence of a person while registering a security code (so-called "Captcha" code) is used
7. after confirming the registration the user can login and logout on the website and get access to his/her member account
8. mostly directly after the registration the user is invited to fill some or all fields in his/her profile including the profile picture

2.2.2 Profile

Every user has his/her personal self-describing site (profile) with some personal information. Users can provide different information about theimself. Following art of personal information can be inserted into the personal profile: general information like name, school, date of birth, contact information (address, ICQ-Number, Skype nickname, telephone number, cell phone number, email address etc.), personal information (relationship status, interests, clubs, music styles preferences, favorite books and movies), information about employment (art and place and other information about employment), current classes (name of instructor, name of class), personal photo (is shown on the personal site). The most of the data is optional. Only some general personal information is mandatory and must be provided by the user during registration.

The user himself can see and edit his personal page. The editing can be executed for parts/blocks of the personal site. Usually the user can change all parts of his personal information, sometimes including even login name and password.

MySpace, Facebook and StudiVZ offer a standard membership for all users. The user of Xing can choose between two different types of membership: "basic" membership and "premium" membership. "Basic" membership offers a bunch of standard functions and services and is free of charge. "Premium" membership offers additional functionality and has to be payed for. In November 2007 5.95 Euro is charged monthly.

A standard profile of Facebook, Myspace or StudiVZ contains following parts:

1. name (showed name, first name, last name)
2. basis information (gender, date of birth, city of living, home town, state, country, ZIP code, race, type of figure, personal goals of membership, political views, religious views)

3. interests and personality (interests, favorite music, movies, books, heroes, sports, hobbies, activities, "about me")

4. relationship (relationship status, sexual orientation, family status, interested in men or women, looking for)

5. schools and education (degree, high school, college, university)

6. lifestyle and background (smoker/not smoker, goals of membership on the site, relationship to alcohol, membership in clubs or unions etc.)

7. companies (name of the company, current or previous employment), city, state, country, position, department, employment date etc.)

8. contact information (email address(es), telephone and cell number(s), address(es), personal web-site(s) etc.)

Since Xing is a business-oriented platform, its profiles look slightly different as profiles of the other three platforms. Since one of the most important goals of the Xing members is to find business partners, job or employees for his/her company, the profile of every member has to present all related professional and personal information about the member. The general form of a Xing profile is a detailed and structured CV.

1. full name, business branch, position, personal photo

2. photo (usually a portrait)

3. personal (search e.g. positions, offer qualifications, interests, organization)

4. professional experience (status: employee or self-employed, organizations and companies, references)

5. education (colleges and universities, qualifications, languages)

6. contact information (private contact information, business contact information, email addresses, instant messaging)

Xing indicates the degree to which the profile is filled out. This should help users to keep track of the state of their profiles. In a business oriented platform like Xing the most of the profile information can be important for the potential employers or business partners.

Users can view their own profile in the form the profile is dispayed to other users, who visits the user's personal site.

Following services are available for personal profiles: the user can

- view and edit his/her personal page
- upload his/her profile photo or replace the existing photo with another one
- determine a group of members of community, who can see details of his/her profile Possible options are:
  1. my friends/contacts
  2. members of my network (e.g. my college or school)
  3. friends of my friends
  4. all registered members
- view personal sites of other people if they belong to the group of users, who is allowed to view it. Otherwise they can see just the basic information about the user e.g. first name and last name of a user and his/her picture
- view his/her own groups or networks
- view a list of his/her friends

Profile plays an important role in all online communities. It helps other community members to build a personality picture of a member. In most cases, user's profile is the first thing what a community members sees when he/she is looking for new contacts. MySpace pays attention to this fact and provides a range of advanced possibilities to organize and to design user's own website.

2.2.3 Friends/Contact List

Every user can search for other members and offer them friendship in scope of the platform. Every user owns a list of friends, members which the user would like to have a closer connection to. A "friend" is a registered member who the user would like permanently communicate with and exchange different data with. In all four platforms every user can suggest friendship to any other user. The other user has to give his/her affirmation for the friendship. Only after an affirmation a friendship is accomplished. A user can deny or accept the friendship offered by another user. In case that the friendship is accepted, both users can see each other in the list of friends on the personal site. This means they have added a new friend to their personal list of friends, which can be viewed and edited only by the owner.

In MySpace and Facebook users are informed about changes in friends profile or some other events e.g. if a friend:

- uploads new photos to his/her profile
- uploads new videos
- adds new blog entry
- makes changes in his/her profile

Building connections to other community members by creating friends' list is not the only way to establish stable connections to certain groups of members. Creating users groups and networks provides the possibility to unite a number of community members using some criteria. The idea of building groups and networks will be stated later on in this chapter.

2.2.4 Photos

In MySpace, Facebook and StudiVZ users can upload an arbitrary number of pictures to their personal site (also called user's space) and group these photos in albums. There is always a single photo which has to be set as the profile photo. In opposite to the three platforms mentioned above, the Xing platform allows every user to upload just one single photo, which will represent the user in his profile. Due to the business character of Xing the functions of the platform are not aimed to entertain its members but just to provide the most important and relevant information about the members.

Following functions are available for users of MySpace, Facebook and StudiVZ:

- upload photos (one or more), create titles for every single photo. Every photo has to belong to an album
- create photo albums, add/delete photos to/from any photo album
- display all existing photo albums as a list with the name and cover picture of every album
- add name and place of shooting for every single photo or album
- set the cover picture of an album
- choose a privacy policy for every single album (accessible only for friends/the owner/all users)

- if a new album is created, it is empty. The user can upload photos and insert them into this new album
- see all photos from all albums
- the user can edit the name and place of an album, privacy policy for a album, titles of photos
- sort the photos within a single album
- move photos from one album into another one
- set a photo as "standard". This means that this photo will be shown as a main profile photo of the user
- use a photo as album cover
- link a photo. The user can link himself and his friends on a photo. This can be deleted later if necessary
- the user can delete a single photo or a complete album

Photos is one of the most usable and popular functions in online communities. The quality of this function plays an important role in the overall popularity of an online community.

2.2.5 Messaging

One of the most important functions of online communities, which is provided by all four examined platforms, is messaging. This function is mandatory for successful social networking systems. The messaging service enables community members to exchange information in form of messages. Messaging is the main form of communication between users. Every user has his own mailbox where he/she can compose,view,send or receive messages. The concept of messaging is very similar in all four platforms. It consists of two different boxes:

1. Inbox

   In Inbox all incoming messages are stored and displayed. They can be marked as seen, unseen or answered. The user can change this attribute by selecting another message status for one, some or all messages in Inbox. All incoming messages are displayed alone with the name and picture of the originator, subject of the message, date and time of receiving/sending, status (seen, unseen, answered). The user can

read, answer or just delete the message.

2. Sent

   In Sent the user can see all messages sent to other users. The messages are displayed in similar way like messages in Inbox. The names and pictures of recipients may be shown. The user can read or delete messages from the box. Also he/she can write another message addressed to one of the recipients. Alternatively, a user can create and send a new message to the user using a link on the recipient's personal site.

   MySpace offers an extended mailbox with following additional functions:

3. New message

   The user can sent messages to his/her friends, which are listed as friends in user's space. A message can be also sent to any other user. In order to do this, the user has to go on the user's personal page and create a message for the certain recipient.

4. Trash

   Deleted messages are stored in "Trash" for a limited time period. After the defined period of time they are completely deleted from the platform.

5. Stored messages

   If the user wants to store some of the existing messages permanently until he/she will remove it from the mailbox by him/herself, then he/she can store it in "Stored messages" box.

6. Friendship requests

   Requests for friendship are messages as well. However, to distinguish them from other messages they are paced into "Friendship requests" box or displayed in a particular way to the user. Every user can accept or deny friendship with a user, who sent a request for friendship.

2.2.6 Groups

"Groups" is a concept which helps to organize the communication with other registered users of the platform. Generally groups are unions of a subsets of online community members. Users can build groups in order to stay in contact with members they share a particular common interest with.

Every user can create his/her own group based on common interest or hobby which is shared by the members of this group. An online community member can create a new group, invite other users to join particular group, search for existing groups, join an existing group. For example the user can create a group of fans of Elvis Presley or a group of members whose hobby is painting. This groups can be viewed as online clubs with a certain character. Every user can belong to multiple groups. If the user is a member of one or more of existing groups, then all these groups are listed on the personal site of the user. There are two general possibilities for a user to become a member of a group:

1. The user can join an existing group

   The examined social networking platforms provide engines for searching groups. All groups are divided into categories e.g. sport, art, campus life, music etc. The user can display all existing groups of one of these categories. The groups are shown in a list with some short information about them (picture, name). Besides, the user can provide a part of the name of the group (keyword) which he/she is looking for (in a certain category or in all categories). If the user has found an appropriate group, he/she can go on the group's page and see following information:

   - name of the group
   - name of the creator of the group
   - category
   - visibility of the group (for all schools/for a single school/for members only etc.)
   - members of the group (names and pictures of the members)

   The user then has the possibility to join this group. The joining can be free (no affirmation by the creator of the group is required) or restricted (the creator of the group can accept or deny a new member). This feature can be set by the creator of the group while registering this group or at any time after that. If a user become a member of a group, he/she can see it in the list of all groups in which he/she is a member on his/her personal site. For every group a participant can choose between following settings:

   - receive a notification email if new group messages has appeared. This option can be turned on or off
   - everybody can see that the user is a member of the group or only members of the group can see the user as a member

If the user is the creator of the group, he/she can change the picture, description, name, category, viewing policy or joining policy of the group. Besides, other users can be invited to join the group by the creator or a group can be completely removed/liquidated by the creator of the group.

2. The user can create a new group and invite other users to join it

   Every registered user can create his/her own group of interest. The creator is automatically the first member of the created group. While creating a new group the user has to:

   - provide a name of the new group
   - choose the corresponding category for the group (depending on specificity of the new group)
   - provide a short description of the group
   - provide the address of the group's homepage (in case that it exists)
   - determine viewing policy. Viewing policy regulates the access to the group's page, especially which community members should have access to it
   - determine the joining policy. The possible options are: every community member is free to join the group or a permission of the creator is required to participate the group
   - provide the title picture of the group

   Members of a certain group can participate in the group's forum. Every single member can create, view and comment the forums entries made by other members.

   The Xing platform restricts the creation of new groups. The creation of a new group on the Xing platform has to be permitted by an administrator. This procedure prevents the creation of non functional and absurd groups.

3. Search for an existing group

   There is a list of categories of groups available on most platforms. Every category contains groups which are characterized to belong to this category. Users can display all group of one of the categories. Aside every category, the number of existing groups is displayed.

   Another way to find a particular group is to use a keyword or to use an advanced search. In advanced search the keyword can be searched as a part of the name of the group or just everywhere in group information. Besides, advanced search options like certain category, country or state can be used while searching.

2.2.7 Networks

Network is a key concept of social networking online platforms. A network can be interpreted as a collection of members connected by one or more relationships. For example, if a user has registered friends then he/she is connected to them by relationship "friend". Friends of a user may have other users as friends which are not in the friends list of the user. In this case the user is connected to them by relationship "friends of my friends". On this way a wide network is created.

Another criteria to join members to a network is a real world organization like university or high school. Student of a certain organization who is a member of the online community build a network. Other users can view the list of this network. This is a good example of how a real world network can be represented within an online community. In Facebook and StudiVZ universities and colleges build the main basis for networks. Being a member of such a network enables users to be aware of events and news in this community and to find real friends within the community.

MySpace offers also one more possibility to find people with similar goals and interests. Every user can register for one or more networks which are divided into categories (e.g. dancing, movie, techniques). Every user can choose a category (e.g. marketing), subcategories (promotion or PR) and position in the branch (e.g. manager). By adding to a certain network a user declares to be interested in issues associated with the categories of network. Now other users can search for people with similar interests or activities.

2.2.8 Search

Users of MySpace, Facebook, StudiVZ and Xing have a range of possibilities for searching. Search is a very important service in the social networking which enables people (users) to find other people they would like to interact with or to find interesting events or other information. Using search new community members can establish contacts to members they are interested in. The arts of search service differs from platform to platform. That is why we will consider search for every examined platform separately.

1. MySpace

   MySpace provides services for searching for friends, colleagues, mates, people with the same interests, music bands, celebrities and other users. The precondition for this search is that the people, which are being searched for, are registered users of MySpace and maintain their profiles.

   The basic search allows the user to search for friends, school mates, members of different networks. The user can search for name, showed name, email address,

name of the school, country, state, network category, position or just keyword.

In advanced search the user can choose some more criteria of search. Here is the list of possible criteria:

- woman, man or both
- age (from/to)
- personal relationship status (single, married, divorced, in a relationship)
- personal goals on MySpace (dating, networks, friends or relationship)
- with a photo only (selectable)
- show names and photos only (selectable)
- country, ZIP code and radius of searching
- attitude towards cigarettes and alcohol
- sexual orientation (heterosexual, homosexual, bisexual)
- education status (school, high school, college etc.)
- religion and children

The user can use all or an arbitrary subset of these criteria for searching. The result of this search is a list of users with pictures and names of the matching users.

2. Facebook

   Facebook offers two kinds of search engines: basic search and advanced search.

   (a) basic search

   In basic search the user can search for people, pages, groups, events and applications (Facebook plugins). The user can also filter results for his network only, any other network (e.g. college or university) or for all networks. Further filters are:

   - male/female/all
   - age (from/to)
   - relationship status (single, married, engaged etc.)
   - interested in men or women
   - looking for dating/friends/a relationship/networking etc.
   - political views and other criteria

(b) advanced search

In this kind of search the user can specify exactly the searching criteria. Following criteria can be used:

- basic info (name, gender, address, political views, religious views etc.)
- contact info (emails, mobile, phone)
- personal info (interests, activities)
- educational and working info

3. StudiVZ

StudiVZ provides following possibilities for searching other registered users: The user can search for a certain person in a certain school e.g. user's own school or in all schools.

Generally following searching operations are provided:

- search for people at user's home school. The user can provide the name or a part of the name of the person, which is being searched for. If no characters were provided, all user's from user's home school are displayed
- search for people at any other (single) school. The school can be picked from a list of existing schools
- group search. This search makes it possible to find different groups of users by entering the name and/or category of the group. A detailed description of this search is given in the next paragraph
- a super-search provides the possibility to look for people using a set of different criteria. An arbitrary combination of following searching criteria can be used:
  - name, birth date, gender
  - home country, ZIP-code, home city, student hostel
  - major, employment status, school
  - personal relationship status
  - political orientation
- a web-search (powered by the Yahoo search machine) is also provided

4. Xing

There are following search engines offered by Xing platform:

- basic search (keyword for first name, last name, branch, city or company)

- advanced search (keyword, name, schools, interests, business information: current position, previous positions, qualifications, address etc.)
- power search (people who viewed my profile, people whose profile I viewed recently, people from my firm or school, my contacts or colleague, the newest members, random member etc.)

As one can see there can be a range of different searching criteria but there is one common characteristic for all four examined platforms: the main goal of their search engines is to enable people to find other people in order to create a contact edge with them.

2.2.9 Settings

There are different setting which can be used in the examined online communities. Settings can help to establish access control to the personal data by other community members or to define the list of users whose messages will be ignored. The character and type of settings vary from system to system and will not be detailed described in this thesis. Generally, we can classify all setting in following groups:

- access control settings
- administration settings

Access control settings enable users to define groups of users who should be able to gain access to user's personal information. However, not only personal information like profile information or personal images can be the object of access control. Online community objects like groups can be also controled by the creator.

Administration setting deal mostly with issues which determine the way certain data should be handled. E.g. a user can set the period of time for which deleted messages remain available on the system. Another example of administration settings is the way messages sent by certain users should be processed. A user can define a list of users whose messages will be ignored. If a user sends a message to another user and is placed on the ignore list of the recipient, the message will not be delivered to the recipient until he/she removes the name of the sender from the list of ignored users.

There are a plenty of different setting, which can be used on the examined platforms. Some of them may be more meaningful than the other ones. The general idea of settings allows users to administrate their activity within the platform.

2.3 Specific Functionality of MySpace

In this section the functions specific for MySpace online community platform are described.

- Bulletins

  A bulletin is a message, which can be posted for all user's friends. Every user can create a bulletin, post it or delete it. A bulletin contains following data: user-creator, date, subject, message-text. A bulletin can be replied, it has an expiration date. Users can view their own bulletins or the ones of other users. Basically, one can see bulletins as regular multicast messages for all friends.

- Design of personal space

  MySpace allows users to design their personal space (personal site) using CSS and HTML. Users can manually edit blocks of their personal site and design it in the way they would like it to be displayed. A useful feature that can be used while designing an own personal site is "diashow". Every user which has one or more photo albums can create a "diashow". Two arts of diashows are possible: diashow for all existing photos or for a certain album. The creation is executed automatically by the system. The created diashows can be included to the personal site and will be displayed to the viewers of the site.

- Calendar

  Calendar is a useful tool which is available for every user. Users can add notes to every day they want. Possible notes are: appointments, events, class event etc. Users can choose different views: days, weeks, month or years. A calendar note contains: title, art of event, date, time, place (street, city, state, country) and a short description of the event. If an event take place periodically, one can notify the calendar about that. Users can choose visibility of their calendars: for all/my friends/ only myself

- Blog

  Every user has his own blog, where he/she can post new messages. New message contains: time, date, subject and category of the blog-message. Other users can comment and evaluate the blog-messages of other users. The owner of the blog can edit/comment or delete his/her blog-messages. Like almost everything in MySpace, users can recommend blogs to other users.

Users of MySpace have a range of possibilities to create their own design of their profiles including blogs. Users can subscribe for blogs of other users. In this case they will be notified if something new happends on the subscribed blog (e.g. new messages or comments are added). Besides, a list of most popular blogs is available for users. In this list the most readable and popular blogs are collected.

- Address book

  Users can store contact information of other users in their address book (personal data, instant messaging information, email addresses, telephone numbers, addresses). Address book entries can be administrated by the owner (edit, delete). Xing offers a similar service. Special features of the address book in Xing will be briefly described later.

2.4 Specific Functionality of Facebook

In this section the functions specific for Facebook online community platform are described.

- Events

  The idea of "Events" function is to inform people about real life events happening and to help to organize efficient and fast new events. An event can be a party, a business meeting or any other occasion.

  All events are divided into three categories: upcoming events, my friend's events and past events. Every user can search for events with a keyword or create a new event by giving event name, tag, network (your school or all users), type of the event, start and end time, and location. An event picture can be added. Finally, Facebook users or other people can be invited to the event.

- Current happening

  Facebook has a lot of services which aim to inform users about current happening related to their friends. For example users can view recent friends photos, recent tagged friends on photos, changes in friend's profiles and much more.

- Plugins

  Facebook provides a possibility to extend the functionality of the platform by adding new functionality. These changes affect only the user's own account. The new functions are grouped to plugins. Plugins are programs which are written by

platform's vendor or by a groups of platform users. In most cases these programs can be used by every member. The available plugins are listed alone with their short description. Users, who included the same type of plugin, can interact with help of it. The type of interaction is specific for every plugin.

Plugins are one of the most important features of Facebook platform. There are about 9800 different plugins available for Facebook users and the number of plugins keeps growing. Every user can edit the list of plugins that he/she would like to use. All plugins are sorted in different categories like music, photo, video, business, dating, education, messaging and many others. The user can search for plugins and add them to his personal page. This plugin structure provides the users and developers a considerable extensibility. New ideas and services can be easily implemented and made available for all users.

2.5 Specific Functionality of StudiVZ

In this section the functions specific for StudiVZ online community platform are described.

1. Message of the day

   An interesting statement from the personal site of one of the users is posted so that every StudiVZ users can read this statement and see the author.

2. Random users

   A random list of users from your network(s) are displayed so that you can see the profile pictures of the users and go to their personal sites.

3. News

   Interesting news for students. On the personal site one can see an area "News", where some announcements are posted. The subject of these announcements can be: opportunities for studying abroad, educational offers, Internet service provider offering their services etc. Every announcement has a hyper link which leads to the information containing website. It can be an internal (placed on the StudiVZ server) or external website, which contains a detailed information about the subject of this particular announcement.

4. Wall/Pinboard

   Every registered user has his own "Pinboard" (or "Wall" in MySpace) on his/her personal site. Other users or the owner himself can place messages on his pinboard.

Everybody, who is allowed to view the personal site of this particular user can also read messages and contribute theirs to the pinboard. Only the owner of the site, which the certain pinboard belongs to, can delete messages posted by other users.

5. Job search

   StudiVZ offers also a search for jobs, internships or trainings. There are a number of announcements alone with a form for searching a new job available for all users. This job and internship search is powered by a partner of StudiVZ, which has a database of available positions in its disposal.

2.6 Specific Functionality of Xing

In this section the functions specific for Xing online community platform are described.

1. Search agent

   Search agent is a particular search which is executed regularly and the result or new search hits are displayed to the user. For example, if a user is looking for an employment with certain conditions, he/she can define a search which will weekly or daily search for jobs with defined criteria and notify the user in case new hits are found.

2. Tags

   Tags are also used in other social networks which are examined in this part. In Xing users can tag other users. With help of tags users can then find information, contained in other user's profiles or just profiles with certain fields values. Tags are words which are associated with an object. A tag describes some characteristics of the object, which can be then searched for.

3. Address book comparison

   Xing provides a synchronization service of address books entries with the address book entries in some wide spread applications like Microsoft Outlook, Microsoft Exchange or Lotus Note. There is a Xing [Xin03] plugin for downloading which is responsible for this synchronization. On this way users do not have to maintain two or more address books in different applications on their computers and on Xing separately.

   Another feature of Xing address book is that users can import their contacts from their email accounts like Googlemail, Hotmail, AOL, Yahoo and others. The user

has just to provide his email address and the password for this email account and Xing will import the contacts from this account automatically.

2.7 New Useful Functions

In this section we introduce a number of new useful functions that can be provided by a social networking platform. These new functions are intended to make online communities more attractive and popular.

1. Multicast messaging

   This function is similar to the "bulletin" concept of MySpace. We remember that a bulletin in MySpace was a message, which can be posted for all user's friends. The idea of multicast messaging should extend this function. The user should have a possibility to send a multicast message to a group of users of the platform. Following options can be given:

   - multicast message to all friends (like "bulletin") or to a subset of them
   - multicast message to all members of a group or to a subset of them
   - multicast message to all members of a network (school, class, university)

   This feature can be useful for informing a larger group of people about happening (e.g. events or announcement) in their social community or network.

2. Group folder

   The concept of groups is implemented in all examined platforms. A useful function that can be implemented in scope of groups function is data sharing with help of a group folder. For example, a group of admirers of Bethhoven music can have a common group folder with mp3 files of Bethhoven's music with expired copyrights claims and may be some videos of classical music concerts in it. The content of this folder should be visible and available for all group members. It should be possible for every group member to upload files to this folder and to access the uploaded files. Only the user who uploaded a certain file should be able to delete this file.

3. Mode of operation

   The main difference between Xing and other three examined social networking platforms is their orientation. As we said earlier, Xing is a business oriented platform aimed to socialize with potential employers, employees and business partners. Facebook, MySpace and StudiVZ are not business oriented and designed

for socializing with fellow students, friends, dates and other people with common interests.

Considering this difference, it must be useful to have different modes of operation:

- basic mode

  In this mode users should be able to communicate with their friends, find new friends, find dates. A list of private friends should be available. All possible non-business features should be present.

- business mode

  In this mode only particular business related information from the profile (user's CV) should be displayed to other users. A list of contacts should contain only business contacts. All other personal issues should not disturb users in this mode.

  This feature can help to merge both sorts of social networks in a single one. For every mode only related profile information and certain functions should be provided.

4. Live chat

   To increase the use of the platform it should be avoided that users have to switch to other applications in order to use some services which are not implemented within the platform. All four examined platforms contain the messaging service but no chat service which is very popular currently. Live chat should provide the possibility to chat with another user which is currently online.

5. File sharing

   The terms "Peer-to-Peer" and "File sharing" have been closely connected in recent years. A Peer-to-Peer architecture of social networking platform is a good precondition for implementing an effective file sharing service. The users of the platform should be able to execute following actions:

   - upload a file to a certain category
   - download any available file from existing categories
   - remove the shared file (executable by the author/uploader)
   - add comments to any shared file

File sharing applications are very popular currently. The merging of functionality of a social networking site and a filesharing application can bring considerable synergy effects related to the usage of the platform.

The legal issues should be considered separately and are out of scope of this thesis. In order to protect the copyrights, the originator/uploader of each shared file should be known (nickname, email address).

2.8 Summary of Functional Analysis

In this section a short summary of functionality analysis is provided. All functions and features described above are classified as mandatory, optional or nice-to-have functions. The classification is made considering the popularity of the functions within examined social networking platforms.

2.8.1 Mandatory Functions

The functions and services which are offered by all examined platforms are considered as mandatory functionality of a social networking platform. Here is a brief list of common services and functions:

1. Registration

   Any Internet user should be able to register and to get his/her login data and account on the platform. An email address and a user password (created by the user) should be the login name and the password for log into the system.

2. Profile

   Every registered user should own his/her own profile with a range of possible personal information including a profile photo. Different profile blocks and fields are described earlier in this thesis.

3. Friends list

   Every user should have a list (can be also empty) of users which stands in relationship as "friend" with the user in scope of the system.

4. Photos

   Users should be able to upload, manage and delete their personal photos or other images on the system. Photos should be organized into albums.

5. Messaging

   Users should be able to send and receive text messages which should be permanently available on the system until the user deletes some or all of his/her messages.

6. Groups

   Users should be able to create new groups, invite other users to join the groups and to join an existing group.

7. Networks

   Users should be able to register themself for different categorized networks of interest (dancing, theater etc.). Universities, schools and organizations, a real member of which the user is, should automatically be the networks of the user.

8. Search

   A powerful search for other users, groups, events and other objects should be provided.

9. Plugins

   With help of plugins new implemented functions can be embedded into the system. On this way the functionality of the platform can be extended.

10. Setting

    Some privacy and other setting should be available for the users.

2.8.2 Optional Functions

In this section optional functions and services of a social networking platform will be described. These are useful functions which do not belong to the core functions of a social networking platform (in opposition to mandatory functions).

1. Live chat

   Live chat enables two users to chat with each other in real time.

2. Events

   Users should be able to create new events, invite other users to participate the events, to see events of other users etc.

3. Wall

   Every user should have an area on his/her personal website where the user and other users can post and view messages.

4. Group folder

   Members of a certain group should be able to share files in common group folder.

5. File sharing

   Users should be able to share files.

2.8.3 Nice-to-have-Functions

In this section some useful functions will be described.

1. Blog

   Blogging has become popular in the last few years. People with completely different social and professional background are using this function to share their experiences and impressions or just to state their opinion to something. Blog is defenetely a useful function which makes an online community more attractive.

2. Calendar

   Calendar helps to organize business and private life of the users. The possibility to share the calendar notes allows people to plan more real life events and to participate them together. Sharing calendar notes means to make it possible for a defined group of user (may be for friends only) to view some personal calendar entries.

3. Multicast messaging

   Multicast messaging is expected to make the user-to-user communication more flexible. An easy way to define a group of people who receives the message would be an appropriate additional function within an online community.

4. Job search

   Access to the job market is a useful function which can be actively used especially by students who is searching for a future employment or an internship.

5. Search agents

   Sometime a certain search have to be executed regularly. A good example for a often executed search is a search for a job or internship. A set of specific criteria for a search can be defined and stored. The user can choose one of such search agents later on and execute it without typing in each time the entire search criteria.

6. Tags

   Tags help to associate data object with certain terms. This function can used both for collective or private use. It is possible to create tags, which will be available for user's own use only or for the usage of all community members.

7. Address book

   Address book holds the contact information of some registered users and enables the owner to interact with these users with low effort.

8. Mode of operation

   We considered two types of online communities: business- and amusement-oriented. It can be reasonable to be able to switch between these two modes of operation. In the business mode an appropriate profile format should be displayed. The access to the system in both modes of operation should be easy, so that the user can easily switch from one mode to another one.

2.9 Summary

In this chapter we examined the most important functions of the four popular online communities: MySpace, Facebook, StudiVZ and Xing. The analysis of the functionality of these online communities showed that a number of functions are commonly offered by all examined communities. There is, however, a set of functions which is typical for only one single community. Providing these functions should make the certain online community more attractive for use. However, the best way to win the majority of the potential users for a certain community is to try to satisfy the various needs of the most of these people. Embedding of popular Peer-to-Peer based applications like filesharing or VoIP telephony, which are currently enormous popular, could help an online community to win more users for the platform. Peer-to-Peer technology appears to be able to provide this functionality for low costs, which brings an advantage for the Peer-to-Peer based online communities in the market competition.

Chapter 3

Peer-to-Peer based Prototype Requirements

In this chapter the functional and technical requirements for a Peer-to-Peer based online community prototype are presented.

3.1 Functional Requirements

In scope of this diploma thesis a prototype of a Peer-to-Peer based online community application was designed and implemented. This prototype is a Java based client. The working name of the prototype is "LifeSocial.KOM" which will be used further in this thesis to refer the prototype.

In this section a detailed functional requirements for LifeSocial.KOM will be provided. The functionality of LifeSocial.KOM is less extensive than functionalities of the existing client/server based social networking platforms since an implementation of a complete and extensive system is not the primary intent of this diploma thesis. The resulting LifeSocial.KOM prototype implements the most of the functions stated in the requirements. Some of them are implemented in a simplified version.

1. Prototype goal definition

   The goal of this software product is unlike the popular online social communities which are using the client/server based technologies, to use the Peer-to-Peer technology to implement the most meaningful and popular functions and services which are offered by platforms like MySpace and StudiVZ. Besides, the architecture of the Peer-to-Peer based prototype is aimed for providing a possibility to easily extent the functionality of the system and to implement any kind of functions which

are considered as useful in scope of an online community platform. Three general challenge are determined for the prototype implementation: design of a Peer-to-Peer based architecture, design of distributed data structures and implementation of the functions, defined in the requirements. The goal of the prototype is to solve all three previously defined challenges and to deliver a working Peer-to-Peer based application. Design and implementation of such Peer-to-Peer prototype qualifies the Peer-to-Peer technology to be able to compete with client/server technology at the example of social networking systems.

2. Application area

   The primary application area is the investigation of the potentials and limitations of a Peer-to-Peer technology for using as basis for implementing an online community platform. The prototype will be used to test the qualities of a Peer-to-Peer based online community and to demonstrate that a competitive online community platform can be built based on Peer-to-Peer technology solutions. Generally, the prototype should be available for every Internet user who is willing to try and to use the functions of this tool. The application will be designed to be started at any personal computer with Internet access and an installed Java Virtual Machine on it.

3. Specification of Functions

   (a) Registration

   For using the full system functionality the user has to be registered in the Peer-to-Peer based social network platform. Every user of the client should be able to create a new user account. Once an account is created it should stay persistent for user's usage. To create a new user account following information have to be provided:

   - user name
   - password
   - email address
   - country of living
   - city of living

   The user name should be proposed by the user. If such user name already exists in the system, the user is notified and requested to provide another user name. The same can be applied to the email address, provided during the registering. Since it is possible to have multiple accounts on the system, it is possible that the user had already registered an account with a certain email

address. Generally an email address can be used only for one user account in scope of LifeSocial.KOM.

With the information given above the user identifies him-/herself as a new user and gets his/her own account on the system. The data entered by the user during the registration should be automatically included into the appropriate fields of the user's profile. Besides, the user participates automatically in the groups, where all members who lives in the given country or city are registered. This means that directly after the registration the user becomes a member of at least two different groups: country and city groups. This should make it easier for new members to build connection to other members.

(b) Logging in/out

Every registered user should be able to log in and log out of the system. With starting the client, the user gets a possibility to enter his/her user name and his/her password to enter the system. After the user passed the log in process he/she should be able to use the full functionality offered by the system. It should be possible to log in using only a single computer. Logging in at another computer beeing already logged in at one computer should result in automatically logging out at the original point.

Further in this section it is assumed that the user has his/her own account and has successfully logged him/her in the system.

(c) Password recovery

In case that the user password is lost, it should be possible to recover the password using the email address or the user name provided while registering. The recovery can be executed by providing either the user name or the email address or both. A new password should be generated and sent to the given email address. It is assumed that the user can get access to the given email account. After the password is recovered, a user should be able to change it. In case the user wants to change the email address, registered in his/her account, a possibility to change the user password should be also provided.

(d) Profile

Every registered user should have his/her personal self-describing site (profile) with some personal information in it. The profile should be accessible for other users of the online community. The profile should be permanently available for all members of the community. The availability of the profile should not depend on the current status of the profile's owner e.g. whether he/she is online or offline. An appropriate storage technique alone with dis-

tributed data structures should solve this problem in scope of a Peer-to-Peer online community.

The profile should contain two different classes of personal information: mandatory information and optional information.

Mandatory information on the user:

- user name
- password
- email address
- country of living
- city of living

Optional information on the user:

- first name and last name
- gender
- date of birth
- personal photo
- interests and personality (interests, favorite music, movies, books, heroes, sports, hobbies, activities, "about me")
- relationship (relationship status, sexual orientation, family status, interested in men or women, looking for)
- lifestyle and background (smoker/not smoker, goals of membership on the site, relationship to alcohol, membership in clubs or unions etc.)
- contact information (email address(es), telephone and cell number(s), ICQ-Number, Skype nickname, address(es), personal website(s) etc.)
- job and career (professional experience, skills, wanted position)
- some other categories...

Personal information which is listed above can be placed, edited, changed or deleted at any time after the user's account creation and before the account is completely deleted. The owner of the profile should be the only one who has the rights for changing his/her profile.

(e) Search (users search)

Users should be able to find other users using the search engine. Search should be designed and implemented as a separate module. The search criteria can be:

- user name of the member

- city of living
- country of living
- email address
- group name
- gender
- some other criteria

The result of the search should be a list of user names. The user names can be then used as entry points to view personal information of the members or to establish communication with them.

(f) Friends

Every user should have a list of other users of the system which is called "friends list". This list can contain zero or more entries. Every entry in the friends' list is a user name of a member and appears only once in the list. This list enables the users to communicate with other users included into this list with low effort. The persons in the list are called "friends" in scope of the system.

Every user should be able to send a "friendship request" to any other user of the system. The other user can accept or deny the friendship. Friendship must be always reflexive, means if user "a" has a user "b" in his/her "friends list" then the user "b" has the user "a" in his/her "friends list". Users should be able also to remove users from the list, what would mean that they terminate the friendship with certain users in scope of the platform.

(g) Messaging

Users should be able to exchange text messages. Having a user name of the recipient, every user should be able to send a text message to the user with the defined user name. Messages should be delivered directly to the recipients if they are online at the moment when the message is sent. Otherwise the messages, which are sent to users who is currently not online, should be stored. The recipient should get access to these messages the next time he/she enters the system. An appropriate distributed data structure for storage within a Peer-to-Peer overlay network should be developed.

Generally every user should own his/her own mailbox which consists of two folders: sent and received messages. Messages, sent or received by the user, should be automatically stored in an appropriate folder. The user should be able to view or delete all messages, stored in his/her own mailbox.

(h) Live chat

Live chat is an instant messaging service, which should simplify the communication of the users. Unlike themessaging function, live chat should be designed only for use by members, who is currently online. Live chat should enable an easy user-to-user communication. During such a communication session text messages should be exchanged. For every conversation with a user a separate interface entity (window) should be established. The user should be able to create text messages which are then sent directly to the opponent. If one of the users leaves the system during a live chat communication, the undelivered messages should not be stored anywhere. The opponent of the user should be notified that his/her opponent is gone.

Messages exchanged during a chat session should be stored in a chat history data structure. The number of message stored in the chat history should be restricted with an appropriate parameter. The most recent chat messages should be stored in the chat history. Chat history entries should be sorted by the user name of the chat opponent. The data object which represents the chat history should be then stored in the network. Only the owner of the chat history should be able to read it.

(i) Groups/Networks

Every user can belong to one or multiple groups. A network is a collection of users joined to an union by a certain criteria e.g. common interest, school, college, university or employment in a common company. Every user should be able to join an existing group or to create a new one. There are some common groups, which can be joined by everyone (city of living or country groups). During the registration the users should be automatically joined to at least two different groups: country group and city of living group. If one of these groups does not exist yet, it should be automatically created by the system. Users should be able to view all groups (as a list of groups' names) of a single user or all members of a certain group. Members of a single group should be able to exchange messages. Messages should be sent to all members of the group.

All groups can have following types in respect to the possibility to join a certain group:

- free (every user of the system can join the network)
- closed (only invited users can join the network)
- restricted (a permission of the creator is needed to join the network)

An additional function in scope of the group concept can be a so-named group folder. All members of a group should be able to share some files in a group folder. The files are contributed by group members and should be permanently available for download. Only users of a certain group should have access to the files stored in the group's folder. The files can be pictures, documents, programs or any related information.

(j) Photos

Every user should be able to upload his/her personal photos or any other photos to the system. All photos should be organize to albums. The user should be able to add/delete photos to/from a certain album. Besides, the user should be able to choose one of the photos stored in a certain album as a album cover. It should be possible for users to view photos and albums of other users. Photos of a certain user can be viewed by all registered users unless the owner restricts the access to the photos e.g. for his/her friends only. An access control based on user name should be provided as additional option. This means that for every album a list of user names can be defined who is allowed to get access to the images.

(k) Current events

Users should be notified about events related to the users which are listed as their friends. There are different types of events the users can be notified about. The possible events can be:

- changes in friend's profile
- changes in friend's photos (e.g. new photos are added)
- changes in friend's groups (e.g. the user joined a new group)

The implementation of this function is optional and was not realized as a function of taken to the function of LifeSocial.KOM.

(l) File Sharing

Every user should be able to share files with all other users. A collection of files which should be shared with other community members have to be stored in a single folder on the local computer. An appropriate search engine should provide a possibility for users to find the needed files. The download of the files, made available by a user, should be possible only for the time the user is online and logged into the system.

(m) Direct File Transfer

Every user should have the possibility to send an arbitrary file to another user. The precondition for the file transmission is that both users (sender and receiver) are online before the file transfer starts and remain online until the whole file is transmitted. Otherwise the transmission cannot be started (e.g. one of the users is offline) or is interrupted and revoked (e.g. one of the users is down during the file transfer) if one of the parties leaves the network.

The most of the functions described in this section were implemented in the LifeSocial.KOM Java-based Peer-to-Peer prototype. Some functions, which deal with access control of certain objects, were not implemented due to absence of an appropriate security and access control concepts.

3.2 Technical Requirements

Peer-to-Peer architecture for online communities should meet following technical requirements:

1. Extendable Architecture

   The architecture should be built in such a way that it can be easily extended by new modules or plugins. System developers or qualified users should be able to extend the functionality of the system by adding/including an arbitrary number of plugins. Each plugin should offer a set of functions, which can be used in scope of the system. It is assumed that the new plugins will be developed and published by the platform's vendor. For the vendor of the platform it should be also possible to exchange certain system components without big development effort. The extendability of the architecture has a considerable impact on the costs invested by the vendor for changing or improving the system. Modularity provides the necessary property for the system to reduce these costs.

2. Communication

   Plugins should be able to communicate over plugin specific messages. The messages should be delivered directly to the recipient if the peer is currently online and should be stored in a well defined way so that the recipient can retrieve undelivered messages after joining the network next time. It is assmumed that plugins of a certain type will communicate with plugins of the same type. However it should be generally possible for plugins to create and sent plugin specific message also to plugins with other types.

3. Storage

   Stored data should be permanently available for all peers. This can be achieved by providing a persistent shared storage. In a Peer-to-Peer network only an efficient distributed storage and replication algorithm can provide this kind of service. Storage should be implemented in such a way that data objects stored by peers should be available for other peers. Appropriate distributed data structures should be designed and provided.

These technical requirements result from the characteristics of the client-server architectures and currently popular online platforms. The entire user data and accounting data is stored there on (a) central server(s). The data is permanently available and can be retrieved and transported to a set of clients. The data can be read or modified by clients or by the vendor. Since there is no central entity in a decentralized Peer-to-Peer network, a reliable persistent storage is required to implement the most of the functions provided by online communities.

Users of a client-server based online platform can communicate over messages or over shared application data. Undelivered messages as well as application data are stored on server(s) and can be retrieved by clients. In a decentralized Peer-to-Peer network the usage of persistent storage is required to store the undelivered messages. If both peers are online, the messages can be routed directly from the sender to the recipient.

Extendability is a very important property of a Peer-to-Peer based architecture since the functions and services offered by currently popular online communities are being continuously improved, changed or extended. To make LifeSocial.KOM to a competitive platform, the new Peer-to-Peer architecture should be plugin-oriented and hence easily expendable.

Chapter 4

Architecture Design

In previous chapters a functionality analysis was accomplished. The functionality of the four most popular online communities was analyzed and described. The functional and technical requirements for a peer-to-peer based architecture were determined and justified. In this chapter a resulting Peer-to-Peer based architecture will be described extensively. After a general overview over the architecture components, a deeper look inside single architecture components and data structures will be given.

4.1 Architecture Overview

In scope of this diploma thesis a model of a Peer-to-Peer based architecture was developed. The developed architecture model is universal and can be used for developing different Peer-to-Peer based systems. The architecture is based on a Peer-to-Peer overlay network. The general idea of the architecture model is based on the concept of service layers.

4.1.1 General Idea

The general idea of the concept of service layers can be formally formulated as follows:

1. The architecture consists of layers, built over each other
2. Each single architecture layer consists of one or more software modules or components
3. Software modules provide clear interface and can be viewed as black boxed software components

4. Each layer provides services for the immediate upper layer and uses services provided by the immediate lower layer

5. The components, located on the same layer, can freely use services of each other using the defined components interface

6. The number of components, a single architecture layer consists of, can be dynamically changed. Components with new functionality can be included on each of the existing architecture layers

7. New architecture layers can be added on the top of the architecture

8. The new architecture layers should provide the same properties in terms of extendability and interface as the existing layers, on top of which they were build on

In the Figure 4.1 you can see the diagram which gives a schematic representation of the service layer architecture model.

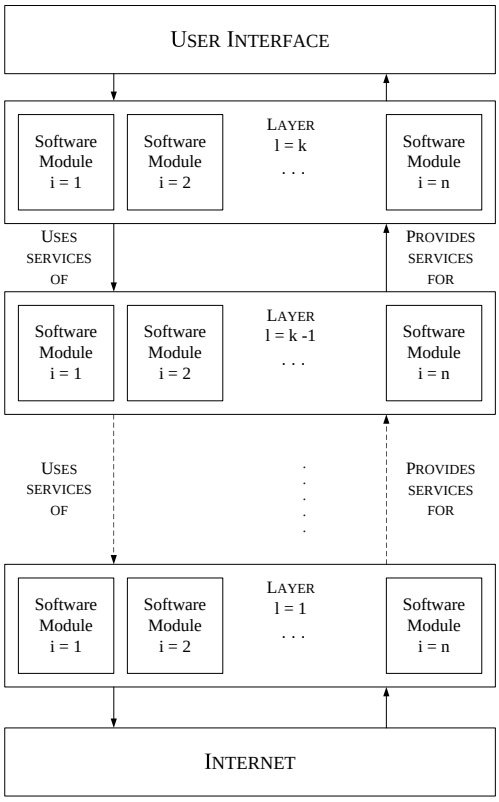


Figure 4.1: Service Layers Model

The upper architecture layer is expected to communicate with the graphical user interface if such interface is required for the application. The lowest layer is the Internet, which serves a basis for a distributed application. The number of layers is not restricted and can be extended on demand. There is, however, a number of layers and components which are considered as mandatory layers and components since they provide the fundamental services for a distributed system. The service layer model is originally designed for a Peer-to-Peer based distributed applications. It is imaginable that this architecture model can be used for distributed application based on other technology.

4.1.2 Peer-to-Peer based architecture

Based on the service layer architecture model described previously, a service oriented Peer-to-Peer architecture was designed. It contains multiple layers, the lowest of which is the Peer-to-Peer overlay network layer. This layer is responsible for building and administrating of the overlay network. The overlay network organizes different participants (peers) of the distributed application. It is assumed that an uniform application client is running on each peer. Usually the overlay network is built in the Internet. The overlay network provides services like creating and administrating of local routing tables, modification and actualization of routing tables, joining and detaching peers to/from the overlay network, routing messages from peer to peer etc. Based on these services other architecture layers are built.

The Figure 4.2 shows the layers and the components of the service oriented Peer-to-Peer architecture. In this section each layer along with included components and their functions will be briefly described.

1. Internet

   Internet is the lowest architecture layer which provides the environment for building a Peer-to-Peer overlay network. It is also called an underlay network. The network infrastructure and the standard communication protocols are provided as services to the immediate upper layer - the Peer-to-Peer overlay network. Using these protocols and services the overlay network builds and administrates the Peer-to-Peer network.

2. Structured Peer-to-Peer Overlay

   Structured Peer-to-Peer overlay layer is responsible for building of a robust, self-organizing overlay network in the Internet and assigning node identifiers to single nodes. Peer-to-Peer overlay layer is responsible for joining and detaching nodes and an effective message routing from node to node using the identifiers assigned to the nodes. Simplified expressed the overlay network connects all users of the

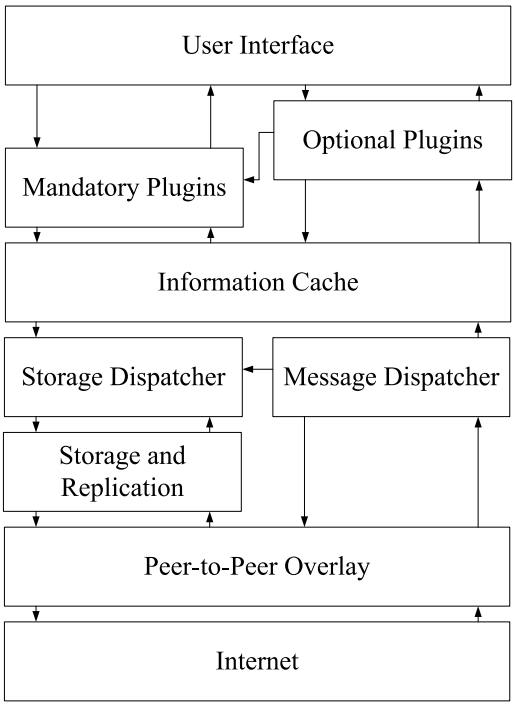


Figure 4.2: Peer-to-Peer Based Architecture

distributed system into one structure and administrates this structure.
It provides following interface for upper layers:

- routeMessage(nodeId, message)

  where the nodeId represents the unique identifier and the message is the data package which has to be delivered to the recipient. The recipient is the one who is responsible for the given identifier.

- connect()

  This method establishes connection to the existing overlay network and creates a new node within the network. If no overlay network exists, this method creates a new one. After the connecting procedure the services of the overlay network can be used.

- disconnect()

  This method detaches the node, the application runs on, from the overlay network.

3. Storage and Replication

   In a Peer-to-Peer network distributed storage algorithms are used to provide a persistent and fail-safe storage service. Such distributed storage service has two important properties: it requires an appropriate data structures for storage of interrelated data objects and it has to provide a possibility for multiple peers to share the stored data. Generally, this kind of storage can be called distributed shared storage because data objects are stored distributed over all peers building the network and the data object can be access by any peer in the network. The storage and replication component of the Peer-to-Peer based architecture (denoted as Storage and Replication on the Figure 4.2) provides the described service for upper architecture layers. This component is responsible for reliable persistent and fail-safe data storage on the Peer-to-Peer overlay network. It is built on top of the Peer-to-Peer overlay network and provides interface for storing and retrieving data objects. In order to make the storage resistant against data loss, a replication strategy is required. Replication increases the data availability and decreases the possibility of data loss. Stored data should be replicated to a set of peers. If one of the nodes holding replicated data leaves the network, the storage component detects this state and initiates a new replication on one of the nodes which is currently alive.

   Following interface is provided by this component:

   - put(object, key)

     Using this method a data object can be stored. The data is associated with the given unique key.

   - get(key)

     This method retrieves a stored data object associated with the given key. If no data object is associated with this key, a 'null' is returned. The key is provided by the upper layer. The key is usually built and provided by a plugin.

   - delete(key)

     This method deletes the data object associated with the given key.

   The properties of this component are resulting from properties of a Peer-to-Peer network and typical client/server based systems. In a client/server based system the data is stored on a central entity which can be accessed by the system users at any point of time. It is important that the data availability in a client/server system does not depend on the number of clients currently using the system. As long as the server is available, the data stored on it is also available for the clients. The

permanent availability of stored data object has to be provided in a Peer-to-Peer based system as well. In most distributed applications it is desired that the data object can be accessed by the peers at any point of time. It can be achieved only with a sophisticated storage algorithm. Since a Peer-to-Peer network is a dynamically changing structure, a replication algorithm is required to make sure that if the nodes which are responsible for certain data objects suddenly leave the network the data objects remain available. To make stored data objects permanently available they should be stored using resources of peers which are currently alive. To avoid data loss after the responsible peer leaves the network, a number of replications of every data objects has to be administrated and permanently adapted according to the current network configuration. If the storage and replication components meets all these requirements, than a storage service similar to a centralized server storage in a client/server architecture is provided for Peer-to-Peer based applications.

4. Message Dispatcher

   A Messaging Dispatcher component is build on top of the overlay network and is responsible for sending and receiving user-to-user messages. This component is designed to work with time persistent account entities. An owner of such system account can join and leave the network. It is assumed that registered users has a unique identifier associated with a certain user account (e.g. user names). These identifiers are used to address application messages. The messages addressed to this certain user account have to be safely delivered to the recipient application. The Message Dispatcher cares about finding the necessary data for direct communication with the recipient independent from the current physical location of the appropriate node within the network. And this is the way how it works:

   (a) an application user enters the system at an arbitrary computer connected to the Internet

   (b) after the log in procedure the system stores user's current data (like IP-address) on the network

   (c) if a message has to be sent from another system client to this particular user, the Message Dispatcher is used

   (d) the Message Dispatcher finds the recipient data stored on the network and uses it to deliver the message

   (e) if the user is currently offline and his/her location within the network is unknown, then the message is stored on the network in appropriate way so that the recipient can receive the message later on

Message Dispatcher uses the basic messaging service provided by the underlying overlay layer.

The interface provided by this component can be described as follows:

- sendMessage(user identifier, message)

  This method sends a message to the user associated with the given user identifier. The message is delivered directly to the recipient if the recipient is currently alive or alternatively stored on the network if the user is not alive. The message can be retrieved by the recipient later on.

- receiveMessage()

  This method is called every time a new message addressed to the current user arrives. The message is then automatically forwarded to the Information Cache layer where it will be held until the upper plugin layer collects it.

The messages should not be understood as a text messages which users of an application could exchange. The message is a data package which holds primary data used by the application. Certainly such system messages can also hold information which will be later interpreted to text or any other art of data. The primary message should be understood as instrument for communication of applications running on different peers in a Peer-to-Peer network. Later on in this section, we will introduce the concept of plugins. In order to get a better understanding of the Message Dispatcher component, the plugin concept will be now briefly described.

Plugins are stand-alone mostly independent high level architecture components. Plugins are intended to communicate with the user interface. They use the most of the services provided by the entire underlying layers. An example of an online community plugin can be a software component which contains a bunch of different functions and provides functionality required for instant messaging service. In the Peer-to-Peer based architecture plugins are expected to implement the core functionality of the application. Plugins running on different network nodes are expected to be able to communicate using plugin specific messages. To identify a certain plugin a message belongs to, a plugin identifier is used. Each message exchanged by applications holds an identifier of the plugin it belongs to.

The account and plugin identifiers, which are used by the Message Dispatcher, can be represented by a number or by a sequence of characters. A message sent with the Message Dispatcher is routed to the user account associated with the given

identifier. Arrived at the right node the message is provided to the plugin with the appropriate plugin identifier.

As it was mentioned above, undelivered messages are stored on the network. The component responsible for storage is Storage Dispatcher. This component is described in the next step.

5. Storage Dispatcher

   Storage Dispatcher is responsible for accessing the shared storage. It is built on top of Storage and Replication component and uses storage and retrieving services provided by this component. Storage Dispatcher is responsible for storing, retrieving and deleting data objects. Every data object implements a common interface and holds a unique storage key (identifier), which is used for storing and retrieving the data object. The storage key is unique and identifies a single data object.

   This component provides following interface:

   - storeItem(item)

     This method stores a data object represented by item. The storage place is determine by the storage key, which is held by each item.

   - lookupItem(storage key)

     This method retrieves the data object associated with the given storage key and returns it as a result. If no data object is associated with this storage key, then the predefined value line 'null' is returned.

   - deleteItem(storage key)

     This method removes the data object associated with the given storage key from the storage capacities.

   The Storage Dispatcher component can be viewed as a black boxed module which accepts data object with certain attributes and provides a reliable and persistent storage service.

   As we could see storage and messaging services are the main kinds of communication in the Peer-to-Peer architecture. Applications running on different nodes can communicate using the shared distributed memory or by sending direct application messages. Both kinds of communication are necessary for the design and implementation of an efficient distributed system. The reasons why we cannot be satisfied by only one of these services will be explained now.

Assume we have only the Message Dispatcher in our disposal. Message Dispatcher allows us to exchange data between users while both the sender and the receiver are online. Otherwise no communication can be established. In most applications it is necessary to be able to send information to users, which are not currently in the network. An online community is one of such applications. This infers that a persistent shared storage is required, which can be accessed by the sender as well as by the recipient of the message.

Further we can assume that only the Storage Dispatcher is available in the examined architecture layer. With help of Storage Dispatcher we can store data object (including plugins messages) and retrieve them independently of the presence of the sender and the receiver in the network. However, many distributed applications like chat or file sharing systems require quick and direct communication between users. The connection in such applications is established once and is used for a certain period of time. During this period of time a lot of messages are exchanged. If we implement this kind of communication using shared memory, it will not be as efficient as sending direct messages from peer to peer. Using shared storage assumes that the data sent to another user is stored on the network and then retrieved by the recipient. The problem is that the recipient has to be notified that new messages are available for him and are stored on the network. Since the only possible communication goes over shared storage the recipient would have to query the shared storage for new messages periodically in order to detect new messages. After detection an additional storage access has to be executed to retrieve the stored message. If we assume that the number of messages can be considerably large, then it is obviously that such technique would result in an overhead of permanent storage accesses. Sending messages directly to the recipient, if it is possible, appears to be more effective and causes less network traffic than the communication over shared storage. However, the last one is also essential for a distributed application.

As we could see it is reasonable not to abstain from any of these modules since a combined utilization of these components provides a flexible and effective communication service which meet requirements for the most Peer-to-Peer application.

6. Information Cache

Information Cache layer consists only of one component which is called with the same name as the architecture layer it belongs to. Information Cache component operates as a data cache for upper architecture layers. Data objects retrieved or stored by Storage Dispatcher as well as messages received by Message Dispatcher are pre-processed and placed into this data cache. The upper architecture layers can

access the data objects stored in this cache. The primary goal of this component is to optimize the data access by an appropriate caching strategy and to make the data objects available for multiple plugins, which are placed in the immediate upper architecture layer. Data objects requested by plugins are looked up from the network with help of Storage Dispatcher and forwarded to the Information Cache where all interested plugins can access the data. Messages, which were addressed to the user, are also held in the Information Cache until the responsible plugin collects them. Upper layers can define the validity period of the data they work with. Information Cache will automatically indicate the requested data object as a cache miss if the validity period of the data object is exceeded. The size of the cache can be explicitly set.

Information Cache offers following interface:

- getItem(storage key)
  This method retrieves the item associated with a given storage key if it is in the cache. If the item is not in the cache, a look up is executed with help of Storage Dispatcher.
- getMessage(plugin identifier)
  This method returns the next message arrived for the certain plugin. An additional method determines whether new unseen messages are available for a certain plugin instance. If no more messages are available, a 'null' value is returned.

The Information Cache makes the lower architecture layers independent from the number and character of the components (plugins) placed in the upper architecture layers. Plugins use the uniform interface to retrieve data object and to send/receive messages. In the following part the advantages of the Information Cache in comparison to an alternative solution are discussed.

There are some alternative strategies to organize communication and event propagation in the considered Peer-to-Peer architecture. The first strategy is to forward all arrived data object requested by certain plugins with help of Storage Dispatcher directly to the requester plugin. The same can be done with messages. They can be immediately forwarded to the recipient plugins. In this case the Storage Dispatcher has to keep track of the requesters which are waiting for a certain data object. The plugins are registered as observers and are notified each time a data object arrives. The message handling by the Message Dispatcher works similar. All plugins have to be registered as observers and are notified each time a new message arrives.

As we mentioned earlier, plugins are intended to be the components which communicate directly with the user interface. Users work with user interface and create events which have to be processed by the plugins. Basically, we can say that the user triggers the work of the plugin. The activity of each plugin should be aimed to process the events and requests provided by the user interface. Since requested data objects arrive with delays, the plugins in the described event and data propagation schema have to react to each arriving data object and message. Plugins have to determine whether the arrived data object, it is being notified about, is still needed to process the user interface requests. This infers a complex algorithm for administrating of notifications coming from Storage Dispatcher and Message Dispatcher. Every single plugin would have to care about this problem on its own way. Besides, the absence of a central caching entity does not prevent the situation when multiple plugins request the same data object and retrieve it multiple times each for itself. Data objects which were issued once and are used multiple times have to be kept within data structures in single plugins which makes the construction of plugins to a complex issue. Another disadvantage of this alternative organization of event propagation is that the Storage and Message Dispatcher components have to be aware of all plugins, acting in the upper layer to be able to notify each of them about the related events.

A better solution is to embed an intermediate layer like Information Cache which would hold the entire data objects and messages inside. This layer is designed as a data cache and is expected to administrate the data objects and messages plugins work with. The advantages of such an Information Cache can be stated as follows:

- plugins do not have to react to irrelevant notifications
- plugins can request data objects and collect them in an appropriate moment
- Storage Dispatcher and Message Dispatcher do not have to be aware of plugins acting in the upper layer. They do not keep track of the plugins instances, but forward all data objects and messages to a single entity - Information Cache. Plugins can get the needed data object directly from the cache
- the whole application is triggered by events happening in the user interface layer. Only plugins whose user interface is currently active are consuming the local memory and CPU resources. Data objects and messages are processed when the plugin becomes active
- data objects do not have to be requested multiple times. If a plugin wants to get access to the object, it just picks this object from the cache

- an appropriate cache strategy reduces the number of requests since the frequently used data objects stay in the cache
- the lower architecture layers are separated from the upper layers and are not affected by changes taking place in the upper architecture layers
- the whole application is not triggered by asynchronous events happening in the lower architecture layer. The entire available data is available in a single entity

All these advantages justify the chosen solution where an intermediate architecture layer called Information Cache is used. We will take a closer look at the design of this component later on in the next chapter.

7. Plugins

   In the Peer-to-Peer based architecture the entire system's functionality visible for the user is divided into single independent applications called plugins. Each such application contains a bunch of interrelated functions which enable the application to provide certain services which can be used mostly independently from other applications. Such applications are referred as "plugins" in this thesis.

   Plugins are located in the upper architecture layer above the Information Cache layer. They are intended to user services of the Information Cache to exchange data with other peers or just to store data persistently in the shared storage. In scope of this thesis, plugins can be defined as stand-alone applications which work with application specific data and offer a clear interface for the upper architecture layer. Plugins are expected to interact directly with user interfaces and to offer services for use to other plugins. In most cases plugins will do both: interact with user interface and provide services for other plugins.

   All Plugins are classified into two groups: mandatory plugins and optional plugins. Plugins whose services are used by other plugins or which belong to the core functionality of the system are classified as mandatory. All other plugins are classified as optional.

   All plugins implement a common interface. The number of plugins included into the system is not restricted.

   The plugin interface contains following methods:

   - getPluginIdentifier()

     This method returns the unique plugin identifier.

- init() and terminate()

  These two methods should contain actions required for starting, initializing or terminating of a single plugin instance.

- pluginMsgsAvailable()

  This method is called to process the arrived plugin specific messages. It contains action which determine the way the messages are handled after arriving.

Mandatory plugins, rather than optional plugins which are dynamically included into the system, are a constant part of the system. Every plugin gets its own unique identifier. The instances of all plugins are managed by a common entity and can be accessed with help of the unique identifier. The common entity which is responsable for providing access to all plugins is called "Plugin Monitor".

Plugin Monitor is software module that is responsible for loading optional plugins into the system when the system is started. The plugins are placed into a predefined directory where the Plugin Monitor can detect and load them dynamically into the system. After loading the optional plugins, all plugins instances can be accessed using Plugin Monitor. As we mentioned above, the number of the plugins is not restricted. A new plugin can be included into the system before the system is started. Developers of new plugins have to care about providing an appropriate user interface in the upper architecture layer. Similar concept of dynamic embedding of software modules into an application is defined in OSGi Service Framework [All07]. Unlike Plugin Monitor, OSGi Service Framework provides functionality for software component management which enables starting and stopping installed components (bundles) or uninstalling components (bundles) within the framework.

The communication between a plugin and a user interface can be described as follows:

- user interface prompts the plugin to provide the specified data
- the plugin prepares and processes the needed data with help of internal functions and the lower architecture layers and makes the data accessable for the interface
- the user interface can prompt the plugin for the prepared data and get the data, which is already available

The user interface components are not triggered by the plugins. The data which is processed within the plugins are made available for the user interface for retrieval.

The plugins concept is aimed to provide a high extendability of the system's functionality. It is easy to extend the functionality by developing of new plugins. The high modularity of the architecture and plugins makes the embedding of new plugins to an easy issue, which does not have a large impact on the rest of the system. It is also possible to build hierarchical structures within the plugins layer which makes a combination of multiple plugins possible. This increases the reuse of the plugins' functions.

8. User Interface

   The User Interface layer provides the necessary interface for the system. This layer consists of multiple components. For each plugin placed in the lower architecture layer an user interface component should be provided. Like other architecture layers this layer can be extended by new components on demand. All plugin specific graphical user interfaces are included in an user interface framework, which enables the individual positioning, loading and presentation of the plugin specific graphical user interfaces.

   An implementation of a graphical user interface was not originally planned as a part of this diploma thesis. The implementation of this layer is a subject of future work of the institute.

4.2 Data Structures

As it was mentioned in the introduction part, every peer in a Peer-to-Peer network contributes its own resources which can be used by all other peers. In this architecture it is assumed that peers make a part of their local memory available for other peers as long as they are part of the network. All these memory resources in a Peer-to-Peer network can be viewed as a shared distributed memory which is available for all network nodes (peers). The Storage Dispatcher component is responsible for the storage of data objects in the network. Every object owns a unique identifier which is used for storing and looking up procedures. The building of these storage keys is in the responsibility of single plugins. To be able to store complex data structures an appropriate distributed data structure is required.

4.2.1 Plugins Communication

Before the developed data structures for the distributed storage in a Peer-to-Peer network are presented, we want to take a look at the way different applications running on different

network nodes communicate with each other.

The entire functionality of an online community platform is implemented by a set of stand-alone plugins. A set of mandatory plugins build the core functionality of the platform. Every plugin operates on it own plugin specific data types. We will now consider an example which demonstrates the communication between applications running on different peers (overlay nodes).

Assume there is an implementation of a Peer-to-Peer based platform, which includes three different plugins: α, β and γ. Assume there are two users, e.g. user A and user B, of the online community which run the application clients on different network nodes. There are two ways how the plugins α, β and γ of the user A and user B can communicate with each other:

1. plugin α of the user A can send a plugin specific message to the plugin α of the user B and vice versa

2. plugin α of the user A can store a plugin specific data object in the network using the shared storage, which is then retrieved by the plugin α of the user B and vice versa

The same works for plugins β and γ. A communication between different types of plugins is also possible. To do that, common data types and message types have to be defined. As we could see, plugins of different application clients running on different network nodes can communicate over direct messages or over the shared memory. Figure 4.3 shows the communication of plugins of type α.

Plugin messages are sent and received over Message Dispatcher. Storing and retrieving data objects is accomplished by Storage Dispatcher. As we already mentioned, undelivered messages have to be stored in case they cannot be delivered directly to the recipient. This means that messages must be stored using shared storage until they are retrieved by the recipient. At this point, we can see that all data objects, plugins deal with, have to provide the same characteristic: all of them must be storable objects. Every object which is stored by Storage Dispatcher holds a unique storage key (identifier). The reason for this is the Distributed Hash Tables (DHT) technology which stands behind the storage algorithm implemented in a Peer-to-Peer network. As it was already mentioned in the introduction part, Distributed Hash Tables work similar to traditional hash tables. Having a key and a value, the value can be efficiently stored in a hash table and then retrieved with minimal effort. The difference of DHT technology is that the entire hash table is split into buckets. Each bucket holds a set of hash table entries and is assigned to exactly one network node. It is possible in a Peer-to-Peer network to find a network node responsible for

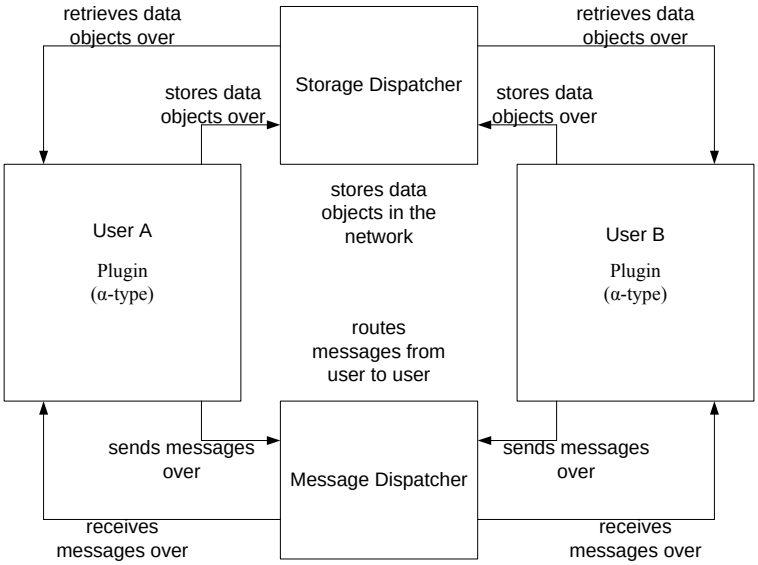


Figure 4.3: Plugin to Plugin Communication

a certain data object and to retrieve this object. This means having a storage key of a data object, we can retrieve, modify and overwrite the associated data object stored with help of DHTs. A storage key can be represented e.g. by a sequence of characters represented by a String object and a standard hash value (e.g. using $SHA-1$). Storage key value is the hash value of the character sequence which is represented with a String object. The character sequence is the argument of the hash function.

Since different plugins operate with different data, we have to provide the possibility to attach header information to the data objects. A header holds the identifier of the plugin the data object belongs to and two more fields: data object identifier and item identifier. These two fields are designed for plugin internal usage. They can be used to determine different types of plugin data or data object role or function.

Every plugin specific data object class extends a common superclass. We will refer this class as 'SharedItem' class. SharedItem has only two class attributes:

1. storage key object
2. header object

The storage key class holds the string and numeric representation of the storage key. Header class holds the data object identifier and item identifier which are represented by

String objects. The instances of classes whose superclass is SharedItem class are automatically storable objects. SharedItem class serves like a container for the data payload, which should be stored. Plugins extend it with the plugin specific payload. Figure 4.4 demonstrates the structure of SharedItem class and a plugin specific data object, which extends it.

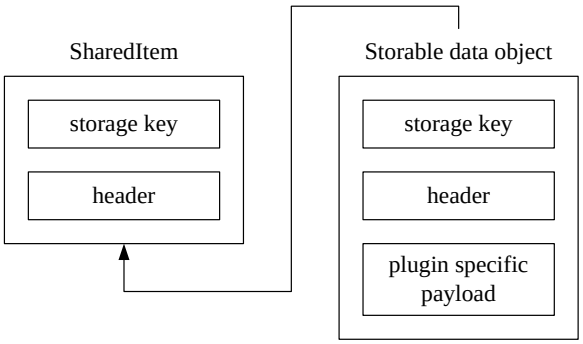


Figure 4.4: SharedItem and Plugin Data Object

4.2.2 Distributed Linked Lists

Plugins data objects are stored using Storage Dispatcher. In order to create a new data object, a new storage key has to be generated. This storage key is unique. A plugin works with a set of classes which represent various data objects which are often coherent. In a client/server based architecture it is relatively easy to store and manage coherent data objects using e.g. relational database systems. In a Peer-to-Peer based architecture a decentralized storage is used. Using distributed hash tables technology, data object are distributed among a set of nodes. Data objects can be accessed with help of their storage keys. To be able to navigate within the distributed storage system and to find the necessary data, we need an appropriate data structure. The data structure which will be now presented is called distributed linked list. The main idea of a distributed linked list can be stated as follows:

1. distributed linked list is a directed graph consisting of a set of storable objects (nodes)
2. all nodes are classified into two groups: entry nodes and internal nodes
3. entry node is a node whose storage key is well known and can be generated as the need arises

4. internal node is a node whose storage key is randomly generated and is previously unknown

5. every node (both entry nodes and internal nodes) can contain a list of storage keys as their payload

6. every storage key refers to a node within the graph, which can hold a list of storage keys and/or other data as its payload

7. a node containing no storage keys of other nodes is called a leaf node

Using this concept, different linked data structures can be built and stored in the network. The structure of distributed linked lists allows to navigate through the shared storage by using the storage keys stored in the objects. Once we built a storage key of an entry node, we can get all data objects which are reachable from this data node.

We will now take a look on an example of a distributed linked list, which will demonstrate a possible application of this data structure.

We consider following situation: every user of an online community owns his/her personal images, which he/she would like to publish within the platform to enable other users and friends to view them. All images are sorted into albums. An album holds some information about the image collection it contains alone with the image collection itself.

Using a distributed liked list, the data structure described above can be stored as follows:

1. every image is stored in a separate object under randomly generated unique storage key

2. every album is also stored in a separate object under randomly generated unique storage key. The payload of every album object is a list of storage keys which refer a set of image objects, which are registered to be in this particular album

3. a special data object holds a list of storage keys which refer a set of albums objects, all of which belong to a single user

4. the storage key of this special data object is well known and can be build by hashing the string representation of the owner's unique user name concatenated with a well known String token (e.g. "albums-list")

Figure 4.5 demonstrates this data structure.

The developers have to make sure that no undesired storage key collision occurs. Within a single system the avoidance of key collisions can be achieved by concatenation

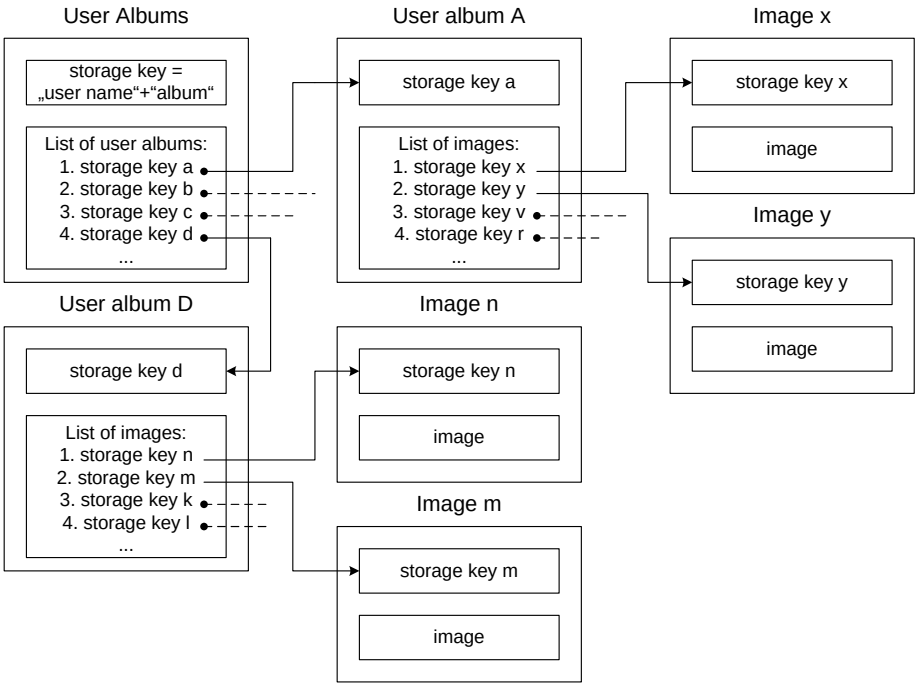


Figure 4.5: Distributed Linked List Example

of well known plugin specific character sequences to the end of the string representation of the storage key. For example the string representation of the plugin identifier can be used in building of storage keys for plugin specific data objects.

This data structure appears as a flexible and simple concept which allows to effectively store and retrieve any kind of complex and coherent data objects. For example a data object can hold storage keys of undelivered messages addressed to a certain user. The list can be modified when new undelivered messages appear or when some of the undelivered messages are delivered and can be removed from the memory.

The concept of distributed linked list data structure is derived from the requirements, which a distributed data structure should meet in order to be able to realize the services and functions provided by a typical client/server based online community platform. The requirements were defined as follows:

- data objects should be placed in a relation to users

  Data objects like user profiles or photos should be associated with a certain user. This is provided by using the user name of the owner in building of the object's storage keys.

- the relation between users and the related data objects has to be unambiguous

  It should be possible to unambiguously determine the owner of the data objects. This is provided by the uniqueness of the user names within the system.

- dynamically generated data objects have to be placed in a relation with users even if the user is offline

  Undelivered messages is an example of dynamically generated data objects. The number of these objects can vary. This requirement is met by holding references to these objects in a data object which is in a relation with the user.

- complex data structures must be efficiently stored

  In Peer-to-Peer based network it is desired to utilize the resources provided by the peers equally. In most situations it is not very efficient to store a large amount of data in one single object. Hence single elements of a complex data object have to be stored separately. Common data objects which holds references to all elements must be available.

Distributed linked list meets all of the requirements and represent a flexible data structure that allows to maintain the data stored in a Peer-to-Peer network in a desired way. Similar to linked list, the data object can contain references to other data objects. The difference to the common linked list data structure is that in the distributed linked lists a single data object can contain references to a arbitrary number of objects within the structure. These references can be easily modified. The resulting data structure looks like a graph where nodes are connected with each other over edges. An important property of distributed liked lists is that in order to be able to navigate through the data structure, certain information about entry data objects have to be owned or constructed. These information should be made available within the system.

4.2.3 Example Scenario of the Architecture

In this section we will examine an example using the data structure described in the example of the previous section. Each step from the start of the application until the termination will be considered in detail in scope of a certain scenario.

First we make following assumptions:

1. a number of peers build a Peer-to-Peer network in the Internet
2. there are two users of the platform which run the application on different network nodes

3. a photo plugin which deals with users' albums and images like described in the previous section is given as an optional plugin and is placed in a predefined directory, where it can be detected and loaded by the Plugin Monitor entity

A user Alice starts her application. A new network node is automatically created and joined to the existing Peer-to-Peer overlay network. As the next step, the optional plugins are loaded into the system from a predefined directory. The photo plugin is loaded and is ready to work. The validity period of the photo plugin data object is set to a certain value. The validity period determines the time slot during which photo plugin data objects available in the Information Cache are considered as valid.

Assume that Alice wants to see images which were published by another user named Robert. Using the graphical user interface Alice requests all albums which belong to Robert. The user interface gives the photos plugin an instruction to load a list of all albums belonging to the user Robert.

The photo plugin builds a storage key for the data object which holds all storage keys of Robert's albums. The storage key has the following structure:

$$\textit{storage key} = \textit{hash}(\textit{(plugin identifier of the photo plugin)} \circ \textit{(user name of the owner)})$$

where ∘ stands for concatenation of words and "hash" for a hash function like SHA-1. Using the built storage key, photo plugin prompts the Information Cache layer for a data object associated with this storage key. Since this data object has not been loaded yet, Information Cache returns a cache miss and initiates the loading of this data object with help of Storage Dispatcher. The Storage Dispatcher retrieves the data object associated with the given storage key from the distributed shared memory and forwards the object to the Information Cache as soon as the data object arrives. The Photo plugin periodically checks the Information Cache for the result and determines when the data object has arrived. The user interface asks the plugin whether the requested data object has arrived. After the list of albums is loaded, it is used by the user interface component to display it to user Alice. Now Alice can see the list of albums belonging to Robert. Assume that Alice decides to view all photos of a certain album from the displayed list. The user interface instructs the plugin to load the list of photos which belong to the certain album. Besides the information about albums, the loaded data object hold all storage keys of Robert's albums. The plugin uses the storage key associated with the album object prompted by the user interface to load it from the shared storage. After the data object is loaded the plugin accesses it in the Information Cache and makes it available for the user interface. Now a list of photos of an album are displayed to user Alice.

Assume that Alice prompts now the user interface to show one of the images from this album. The user interface instructs the plugins and the loading of the image starts. While the object is being loaded, Alice decides that she would rather see another image from this album and prompts the user interface to show it. The user interface instructs the plugin to load the image and checks periodically for the result. In between, the previously requested image arrives into the Information Cache. Since there is no notification from the Information Cache layer, the photo plugin is not busy with processing of this currently irrelevant object. It checks the Information Cache for the currently awaited image, upon the user interface request, and makes it available for the user interface. On this way, the overhead caused by the analyzing of irrelevant data object is avoided.

Now we assume that the photo plugin is designed in such a way that all images of a single album can be loaded at once if the user interface provides an appropriate instruction. Assume, Alice prompts the user interface to show all images of a certain album. The user interface instructs the photo plugin to load all images of an album. The plugin initializes the loading of all of the requested data objects. The plugin picks periodically the data object available in the Information Cache by now. The data objects are then processed and made available for the user interface. Plugin keeps track of the state of loading completion and indicates passively the finish of the loading to the user interface. Means, the user interface can ask whether the collection of data objects, loaded by now and made available for the presentation, is completely loaded or not.

If the number of images is considerably large, it can happen that some of the images arrive earlier than the others. The plugin periodically checks the Information Cache upon the user interface request for the awaited images and makes all images loaded so far available for the user interface. The user interface prompts the plugin for results available so far and presents them to the user Alice.

Now assume that while loading a set of images which belong to an album the plugin is instructed to load a set of images which belong to another album. While loading the new set of images the previously requested images arrive into the Information Cache. If the plugin would be notified about all these irrelevant events, it would result in a huge processing overhead within the plugin or within the user interface component which would have to determine whether the image is relevant or not for this point of time.

Figure 4.6 shows a diagram of architecture layers communication.

On the diagram we can see that the photo plugin acts as a cache for specific data objects. The photo plugin gets the object from Information Cache and makes them available for the user interface. The user interface instructs the photo plugin about the data object it should pre-process for it. The Information Cache serves as a big cache which is used by all plugins. Information Cache makes a number of recently used or requested data objects

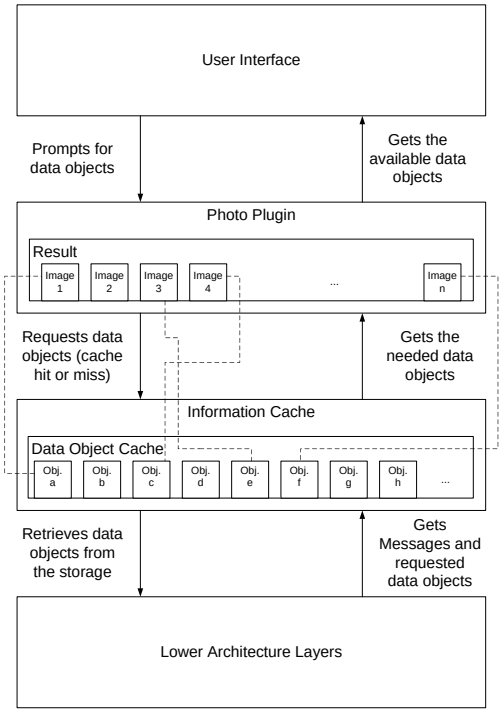


Figure 4.6: Layers Communication Example

available for use. This kind of communication avoids an overhead that would be caused by notifications of upper layers by the lower layers and makes the architecture layer highly independent from each other.

Chapter 5

Prototype Design and Implementation

As it was stated in previous chapters, a Peer-to-Peer based prototype of an online community platform is implemented in scope of this diploma thesis. The design and implementation of this prototype is primary aimed to show the general possibility to implement typical functionality and services provided by client/server based systems in a context of Peer-to-Peer based system. In this section the implemented prototype with the project name "LifeSocial.KOM" will be presented. We will first meet the components used in the Peer-to-Peer based architecture model presented in the previous chapter and their concrete implementation. Then we go over to the implementation details of LifeSocial.KOM plugins, which implement the functions described in the Chapter 3.

5.1 Architecture Components

The Peer-to-Peer architecture model is used as a basis architecture for the LifeSocial.KOM prototype. At the place of abstract layers and components the real implementations based on a Peer-to-Peer overlay network schemata are used. In Figure 5.1 you can see the diagram which presents the resulting LifeSocial.KOM architecture.

The architecture includes the same layers and components as the Peer-to-Peer architecture model presented in the previous chapter. For the overlay network layer an implementation of a Pastry [RD01a] routing scheme FreePastry [DEEH$^+$02] is used. The functionality of the storage and replication component is provided by PAST [RD01b] utility, a software tool created on the basis of FreePastry [DEEH$^+$02]. The other layers and components were implemented in scope of this diploma thesis. We will start with the lowest layer - the Peer-to-Peer overlay and will take a look on all included layers and

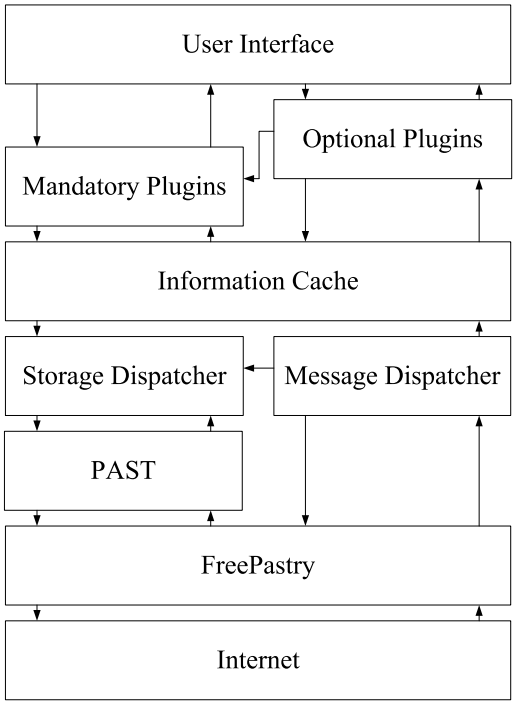


Figure 5.1: Applying Peer-to-Peer Architecture

components.

The Plugins Layer (Mandatory Plugins and Optional Plugins components) consists of multiple plugins. The implementation of LifeSocial.KOM includes seven mandatory and one optional plugin. The mandatory plugins belong to the core functionality of LifecSocial.KOM. The optional plugin can be included on demand.

The Figure 5.2 demonstrates the Plugin Layer structure of LifeSocial.KOM. All plugins except the Live Chat plugin are mandatory plugins. For every plugin a user interface is implemented and included into a common framework for user interfaces. Each plugin interacts with its own user interface. All plugins can interact directly with the Information Cache. Some plugins use services of other plugins. This relation between plugins are called plugin dependencies in scope of this thesis. In Figure 5.2 some plugin dependencies are shown. These dependencies are marked with letters α, β, γ and δ. The dependencies denote following relation between plugins:

α - Login plugin instructs the Profile plugin to create a new user profile. Login plugin provides the necessary data for creation of a new user profile

β - while creating a new profile or editing an existing profile, Profile plugin instructs

Groups plugin to adjust the appropriate groups. For example if a user changes the city of living in his/her profile, then it results in leaving the group associated with the old city of living and joining the group associated with the new city of living

γ - Profile plugin registers the criteria associated with a certain user by providing appropriate instructions to the Search plugin. The information contained in a user profile is then used by Search plugin for searching the users associated with this information

δ - Login plugin instructs the Search plugin to register e.g. the user name and the email address associated with a certain user as a criteria, using which this user can be found

New plugins can create relationships to the mandatory plugins by building new dependencies. For example a new plugin can use services of Search plugin to register new cirteria, which is then associated with a certain user and can be used for search of this user.

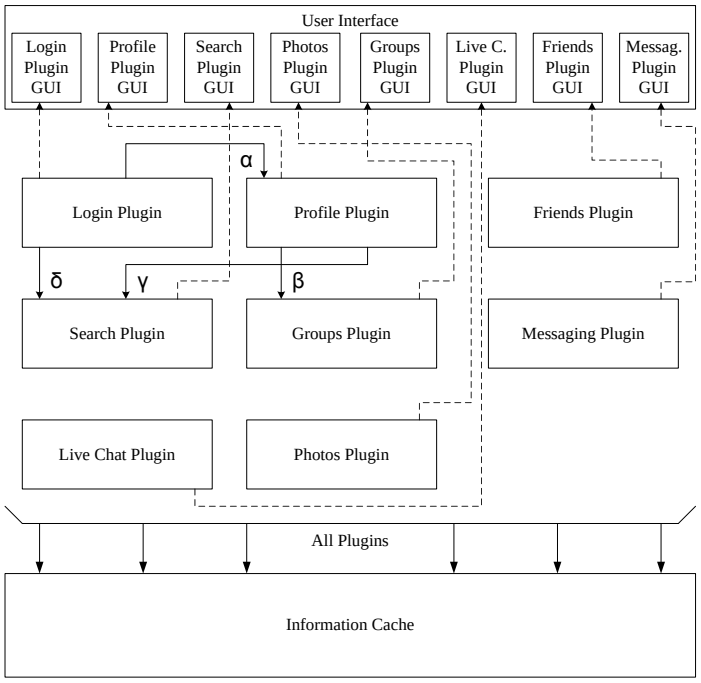


Figure 5.2: Plugins Layer

We will consider the implementation details of single plugins further in this chapter.

5.1.1 Pastry

Pastry is generic routing and location scheme which is supposed to be used in Peer-to-Peer networks. Pastry nodes form a completely decentralized, structured Peer-to-Peer overlay network within the Internet. Pastry was designed as a substrate for high scalable and generic overlay network, which is supposed to be used as basis for building Peer-to-Peer application [RD01a]. The main properties of Pastry are: decentralization, self-organization and fault-tolerance as far this allows the quality of Internet connections [DEEH$^+$02].

Every node in the Pastry overlay network substrate gets a unique numeric identifier assigned. The identifier is called 'nodeId' and is generated randomly for each node. The values of 'nodeIds' are uniformly distributed over the numeric space the identifiers are picked from. In Pastry the 'nodeIds' are 128bit values [RD01a].

Except assigning the node identifiers and organizing an overlay network, Pastry offers an efficient routing functionality. Given a numeric value in the 128bit numeric space and a message, Pastry efficiently routes this message to the network node, which has an identifier numerically closest to the given numeric value [RD01a]. For this thesis, we will simplify this function and will describe it as follows: given a message and a nodeId, Pastry efficiently routes the messages to the node responsible for this Id. Pastry provides an efficient routing algorithm which guaranties that the message will be delivered to the recipient node in O(log N) steps, where N denotes the number of nodes participating the overlay [RD01a]. The routing algorithm requires that each network node keeps track of its immediate neighbors. The neighbors are defined as nodes of the overlay network with the identifiers numerically closest to the regarded node's identifier [RD01a]. The applications build on top of Pastry are notified if a neighbor node leaves/joins the network or a network failure occurs [RD01a].

Pastry belongs to the class of structured Peer-to-Peer systems. Pastry is not the only one of its art. Further Structured Peer-to-Peer systems are Kademnia [MM], Chord [SMK$^+$01], CAN [RFH$^+$01], Tapestry [ZJ01] and P-Grid [Abe01]. All these systems have one in common: all of them implement a DHT as a basis for network administration. Pastry [RD01a] is the most supported project from all of them. The open source implementation of Pastry - FreePastry [DEEH$^+$02] is well documented and maintained Peer-to-Peer system. A good supporting infrastructure including tutorials and mailing list discussion moderated by the developers makes FreePastry [DEEH$^+$02] to an attractive implementation for research.

5.1.2 FreePastry

There exist a number of implementations of Pastry which are currently available as open source tools. Two of them are FreePastry [DEEH+02] from Rice University and SimPastry/VisPastry [Res02] from Microsoft Research. Both FreePastry [DEEH+02] and SimPastry/VisPastry [Res02] support a common API. This means that both provide am identical set of software calls and routines, which is defined by a common entity. This makes them to exchangeable components. Any other DHT implementation can be used at the place of FreePastry. Since FreePastry is an open source implementation of Pastry, it was considered as a suitable basis for a Peer-to-Peer based architecture for online communities. FreePastry was originally designed and implemented to allow researches evaluate the behavior and properties of Pastry.

FreePastry was also intended as platform for application development in Peer-to-Peer substrates. In order to have a better understanding of FreePastry functionality, we will take a short look at a simplified Pastry application programming interface, which is provided by FreePastry [DEEH+02]:

- every node has a unique 128bit identifier ('nodeId') [RD01a]
- route(msg, nodeId) causes FreePastry to route the given message to the node responsible for the specified 'nodeId'
- deliver(msg, key) is automatically called when the message arrives on the recipient side

FreePastry serves as a basis for LifeSocial.KOM implementation. In the selection of a Peer-to-Peer overlay network implementation, FreePastry was favored over other open source implementation like JXTA [Mic01] by reason that FreePastry is a well documented and intensively supported implementation with a number of detailed tutorials and examples, which help to reduce the development time and effort. Further, FreePastry's functionality is identical to a DHT. No unnecessary features are implemented, which would increase the complexity. Besides, a number of different research projects were started on the basis of Pastry and some useful Peer-to-Peer based tools like Scribe[CDKR02] and SplitStream [CDK+03] are already implemented on top of FreePastry and are mostly available as open source implementations.

5.1.3 PAST

As described in the technical requirements of the resulting Peer-to-Peer based online community software prototype, a large scale Peer-to-Peer storage utility is required to be able

to provide the most of the mandatory functionality of online communities. PAST is a storage management and replication tool based on FreePastry [DEEH+02]. It provides persistent storage, caching and replication of data among Pastry nodes [RD01b]. Files in PAST are assigned uniformly distributed identifiers and replicas of a file are stored at nodes whose identifiers are numerically closest to the file's identifier in the DHT [RD01b].

In scope of this work, PAST is used as a storage and replication utility. Given an object S and a unique identifier Id, which is associated with this object, PAST provides following application programming interface:

- store(S, Id)

  causes PAST to store and replicate S across multiple Pastry nodes

- lookup(Id)

  causes PAST to retrieve the object S from one of the nodes where the object is stored at

PAST offers an efficient replication functionality which ensures strong persistence and high availability of data stored with PAST [RD01b]. The data objects stored with PAST are replicated to a set of Pastry nodes. If one of these nodes leaves the network, a new replication of the data object is created on one of the nodes. The number of replications and the time intervals for replication maintenance procedures can be set by changing the system's parameters.

In scope of this diploma thesis, a number of additional requirements for an storage and replication component were established. Data objects, an online community operates on, can be rather large (e.g. images). Network transfer of such data is expensive in respect to traffic and bandwidth. PAST storage utility offers lookup operation to get the needed data object. If a node executes a lookup request, PAST searches for the node responsible for the requested data object and transfers it to the requester. Sometimes however it is redundant to transmit the whole object over the network. If a data object has to be modified, it is enough to send appropriate information to the node which is holding the data object. The node, which is responsible for this information, gets the update information and modifies the data object locally. With other words, in order to avoid redundant traffic in the overlay network it should be possible to modify data remotely. Besides storage and lookup functionality, a storage and replication component should provide following functionality:

- mutable data objects

  The PAST storage utility is originally designed for operating on immutable data object. Once an object is stored with PAST, it cannot be altered. Mutability of data object is necessary for most applications. Mutability is a prerequisite for an effectiveness of the storage and replication component in scope of the Peer-to-Peer architecture.

  This requirement was met in this thesis by extending of a PAST class, which represents the storable data object (PastContent). It was possible to change the type of the objects to "mutable" which enabled the PAST utility to overwrite the content if it was required. However, the PAST utility is not originally designed for handling mutable data objects. The replication algorithm used in PAST does not guarantee the consistency of all replicas created by PAST if the original data object is altered. Providing consistency of the data is a part of future work.

- update(storage key, update information object)

  This function modifies the data object associated with the given storage key. The information about the way the data object is to be modified is contained in the update information object. The updates should be executed remotely without transferring the data to the initiating node.

  This requirement was met by implementing of a special procedure for updating data objects. The whole data object was transferred to a peer, which was the initiator of the update, then it was updated and stored back in the network. The copy of the original data object was overwritten during the storing. This kind of update is less efficient than an update executed remotely since additional bandwidth and processing overhead arise, but the solution was satisfying in scope of this thesis.

- remove(storage key)

  This function removes the data object associated with the given storage key. The data object is removed remotely like described above. The PAST storage utility is not designed for deleting stored objects. Once an object is stored with PAST, it cannot be deleted by PAST. Data objects which are not used any more stress the storage capacities and cause redundant network traffic.

  In order to implement this function, a special solution using PAST was designed. Data objects which have to be removed are replaced with an data object which does not carry any payload and hence is minimal in respect to storage utilization. This solution is satisfying for this thesis but should be improved in the future.

The data modifications executed by these two functions should have effect on all replicas of the modified data object. It is important that the states of data objects stored by the storage and replication component remain consistent even after some of the nodes, which are holding the modified data object, leave the network. An algorithm which cares about data consistency is needed and is subject of future work.

In a storage system where data objects are accessed by multiple entities simultaneously an access control system has to be provided. The access control system should regulate following issues:

- data modification should be executed only if a certain permission is available
- the access to data objects should be restricted to read or write permissions or both
- concurrent data modification have to be resolved
- the modifications on data objects have to be propagated to all replicas. The consistency of data has to be guaranteed

PAST storage utility does not provide any data access control mechanism, which should regulate the permissions for data modification and removal. Each node in the network is authorized to store and lookup data objects with help of the PAST utility. The component should provide the data access control functionality described above which allows to assign and modify different access and modification permissions to created data objects. Generally, the creator of an object should decide about these permissions.

5.1.4 Storage Dispatcher and Message Dispatcher

As described in the previous section, the Storage Dispatcher provides storage services for platform specific data objects. All data objects and messages in LifeSocial.KOM extend the common class SharedItem. This class (SharedItem) contains attributes representing storage key and header of the data object and supplies a data object with properties which make it to a storable object. Given a storage key and a data object associated with this key, Storage Dispatcher performs efficient storage, retrieval or removal operations on this data object. When initialized, Storage Dispatcher reserves a certain amount of hard disk memory on the local machine, which is then used by Storage Dispatcher. This memory is the contribution of the peer to the overall network resources available for all peers. The amount of memory which is reserved by the system can be set centralized by changing a appropriate system property. It is assumed that this value is set by the vendor of the software to force the peers to contribute an equal amount of resources to share.

Message Dispatcher provides user-to-user message communication. Given a user name, which is unique for all users within LifeSocial.KOM, and a message, Message Dispatcher effectively routes the message to the receiver. Messages sent by Message Dispatcher can be delivered directly to a certain user or stored using Storage Dispatcher. The stored messages can be retrieved by the receiver next time he/she uses the system. All messages extend the SharedItem class and hence provide the same properties as storable objects. Each message contains a storage key and a header. The storage key is used when the message has to be stored in the network and cannot be delivered directly to the recipient. Besides, messages provide certain properties which make them to sendable objects. In addition to functions which belong to the user interface of a Message Dispatcher component, this concrete implementation of Message Dispatcher provides an additional function:

- getUndeliveredMessages(pluginId)

Given a plugin identifier (pluginId), Message Dispatcher checks if any new undelivered messages are available in the shared system storage for this certain plugin and the user currently logged in on the system. The new messages are retrieved from the storage and then removed from the network.

Plugins are free to decide when this function is executed. Mostly, it is called when the system is started and a plugin is initialized.

5.1.5 Information Cache

Information Cache contains two data structures for administrating of data. There are two types of objects the Information Cache component deals with:

- messages
- data objects

Messages and data objects are stored in separate data structures within the Information Cache. Plugins can access data objects or messages in the cache over the defined interface.

As we already mentioned, every message belongs to a certain plugin within the system and holds an appropriate plugin identifier. All messages, which are delivered and placed into the Information Cache are held in a hash table where the key of an entry is the plugin identifier the messages belong to. In order to be able to store multiple messages in only one hash table entry, a collection of messages is stored as a value associated with every single key. Every plugin identifier in the hash table is mapped to a collection which holds all unseen messages for this plugin. The collections are implemented as queues so that

messages are put into an appropriate queue in the order they arrive at the recipient. After a plugin retrieves a message from the cache it is automatically removed from the cache. Plugins can check if new messages are available using a certain method provided by the Information Cache.

Data objects are held also in a hash table structure where the storage key of the data object is used as a key of the hash table entry. The usage of the storage key as an entry key is justified by the uniqueness of all storage keys. The size of the data object cache is determined by the number of data objects contained in the cache. The size can be restricted by setting one of the system properties. To increase the number of cache hits the "last recently used" caching strategy is used for administrating data object inside the cache. Recently referenced data objects are placed to the former part of the cache. If the number of data objects stored in the cache exceeds the maximal cache size, then data objects placed at the end of the cache are removed. On this way, frequently referenced data objects are permanently available in the cache.

The Information Cache plays an important role in the Peer-to-Peer based architecture. It accumulates all data object the application deals with and offers a clear interface for upper layers for accessing the data in the cache. The primary function of Information Cache is to supply upper layers like plugins with the required data. Incoming data objects and messages are not propagated to single plugins but stay in Information Cache where plugins can access them. As it was mentioned in the Chapter 4, the advantage of this technique is that plugins do not have to react to asynchronous events happening in the network. Plugins can decide what data object they want to have at each point of time and get it from the cache. With help of Information Cache plugins can combine multiple data objects from the cache to a data package. The plugins are not triggered by incoming data objects or messages but fetch the available data from the cache.

Some plugin specific data packages can be a combination of a set of single data objects. Single data objects are stored separately in the network which means that they can be physically located on different network nodes. In order to retrieve such data package from the storage, all single data objects have to be retrieved using Storage Dispatcher. Some of the requested objects can be delayed and arrive later then the others. In this case is it appropriate to provide a possibility which indicates the state of retrieval for every single data object requested by a plugin instance. This helps plugins to determine in what state the retrieval of data objects currently remains. This mechanism is implemented in Information Cache by using of a container class named ItemContainer. This container class holds a reference to the requested data object and a state of the retrieval process.

The state can have three different values which are:

1. requested object is expected
2. requested object does not exist
3. requested object is completely loaded

These states indicate whether the object is already in the cache or is being loaded from a network node. In case no data object associated with the given storage key exists, the state of the container class will be set to an appropriate value. This feature allows plugin instances which work with combined data packages to determine whether every single part of the data package has been loaded. This concept helps plugins to provide information for the user interface which indicates the status of the data processing.

5.1.6 Assumptions

In scope of this diploma thesis some design related assumptions were made. The most of these assumptions are related to issues which were not addressed in this work but are extremely important for the entire concept of building Peer-to-Peer based systems based on a service oriented architecture model.

While building the Peer-to-Peer architecture and implementing the Peer-to-Peer based prototype following assumptions were made:

1. the Peer-to-Peer overlay network implementation provides reliable functionality for building a stable and fault tolerant overlay network in the Internet, dynamical joining and detaching of nodes and effective and reliable routing of messages from node to node
2. the storage and replication component provides permanent, fail-safe and reliable storage functionality. The stored data is administrated by the utility in such a way that data consistency and availability are guaranteed
3. the storage and replication component provides a mechanism to synchronize the data access
4. the storage and replication component provides functionality for an effective storing, retrieving and modifying of data. This includes the possibility to modify the stored data remotely
5. security is provided in different components of the architecture. The authenticity of users and data, integrity and privacy of data are given

Unfortunately the implementations of different components which are used in the Peer-to-Peer architecture for LifeSocial.KOM do not meet all the requirements that result from the assumptions given above. It is expected that the most of the requirements will be addressed in future works of the institute.

5.1.7 Security issues

Security in a Peer-to-Peer system is a complicated issue and has not been treated in this diploma thesis. This topic will be handled in future works of the institute. It will include authentication procedure for users, access control system for shared distributed storage, encryption of data payload, secure, authenticated communication between peers and other security related concepts. In [Wal03] a number of security issues including secure assignment of node identifiers, ejecting misbehaving nodes, self-certifying data and resource utilization fairness are discussed. In this diploma thesis we assume that security will be included later as an architecture layer or/and as additional components in different architecture layers.

5.2 Online Community Prototype Design

LifeSocial.KOM is a Java based prototype of a Peer-to-Peer online community platform. The decision to use Java programming language for implementing LifeSocial.KOM was justified by the FreePastry implementation which is also a Java based application and is written in Java programming language. Besides, Java technology is currently one of the most supported technologies. Many mobile devices can execute Java applications. Further, Java applications can be converted to web-based applications with low effort.

Before the implemented plugins are considered we want to take a look at the design of LifeSocial.KOM. The general design properties of LifeSocial.KOM implementation are stated as follows:

1. every application runs on a single Pastry node
2. the node and a node identifier is generated and assigned every time the application is started
3. after the node is generated, it is automatically connected to the network
4. every node is associated with a data object containing the IP address of the node (NodeHandle)

5. every user gets a unique user name assigned while registering. This user name is persistent and is associated with a certain user account. The user name identifies the user within the system

6. every plugin included into the system has a unique plugin identifier, which is represented by a sequence of characters

7. all plugins are registered in the system using a common plugin administration object called Plugin Monitor. All plugins can be accessed using their plugin identifier over the Plugin Monitor instance

8. the functionality of the system is provided by a set of applications called plugins. A number of plugins provide the core functionality of the application and must be included into the system. Plugins which do not belong to the core functionality can be included into the system dynamically while starting and initializing the system. Plugin Monitor is responsible for loading the plugins

9. every plugin can communicate only with its own plugin specific graphical user interface (GUI). The GUI components should be included into a common extendable structure. New GUI components should be easily included into the system when a new plugin is included. GUI is not implemented in scope of this diploma thesis and is a subject of future work. However, a text-based user interface was implemented for LifeSocial.KOM

10. two different LifeSocial.KOM clients running on different network nodes can communicate over messages and/or over the shared network memory. All messages are plugin specific and contain plugin specific payload

11. each plugin recieves only messages of its own type. Knowing the type of a plugin message, other plugins can generate and send messages to this plugin

12. undelivered plugin messages are stored and can be retrieved by the recipient at an appropriate point of time

13. distributed linked lists are used as a basis data structure for storing data objects

14. it is only possible to be logged into the system at one computer. Logging in at another computer results in logging out of the user at the former one

15. some of the profile fields are used automatically as groups. For example, if a user lives in Germany then he/she is automatically a member of the 'Germany' group.

This feature contributes to the concept of social networking where networks are build using a certain connecting criteria

Other design decisions for LifeSocial.KOM will be stated later in this section when the description of single plugins will be given.

5.3 LiveSocial.KOM Plugins

Plugins are stand-alone applications which can be mostly used independently of other plugins. However, there are plugins which belong to core functionality of the application (e.g. a plugin responsible for user account management) and their correct execution is the prerequisite for execution of other plugin. A plugin sends and receives plugin specific messages, creates and stores plugin specific data objects, retrieves data object from the shared memory. Plugins are expected to interact directly with the user interface. To each plugin a corresponding user interface has to be designed for this special type of plugin.

LifeSocial.KOM includes a number of plugins implemented in scope of this diploma thesis. Every plugin implements a set of functions which are aimed to provide one of the functions described in the prototype requirements in Chapter 3.

In this section a simplified description of the plugins will be given. An extensive description is out of scope of this thesis. For more detailed information, the LifeSocial.KOM documentation can be used.

5.3.1 Login Plugin

Login plugin belongs to the core part of LifeSocial.KOM and is responsible for creation and administration of user accounts. It provides functions for logging into the system, recovering and changing the password and logging out of the system. In order to be able to use the system, every user has to create an user account. Once a user account is created, it enables the user to use the system until the account is deleted. A user of LifeSocial.KOM can create multiple accounts but it is expected that only one single account will be used in most cases.

Generally, following functions are provided by the Login Plugin:

1. log in
2. log out
3. create a new user account
4. change email address

5. change password

6. recover password over user name or email address

Login Plugin is the plugin which is initialized first when the system is started. Other plugins are initialized after the user logs into the system by providing his/her user name and the corresponding password. Optional plugins are bound into the system and registered at a central plugin administration entity (Plugin Monitor) alone with mandatory plugins. After the initializing, the functionality of all plugins is available for the user.

Login plugin works with data objects represented by LoginItem class. This class extends the SharedItem class which provides the storable functionality for the object and extends it with a payload. Each LoginItem stored in the network holds the entire account information for exactly one user account. The LoginItem can be retrieved using an appropriate storage key which is built using the user name. The storage key is built as follows:

$$\textit{storage key} = \textit{hash}(\textit{user name} \circ \textit{Login Plugin identifier} \circ \textit{String token})$$

where the "user name" denotes the String representation of the user name, "Login Plugin identifier" denotes the String representation of the unique Login Plugin identifier, the "String token" stands for Login Plugin specific String token, which is used to avoid key collisions and $\circ$ stand for concatenation operations.

An example will demonstrate how the storage key for a Login Item data object is built. Assume that

- the user name is "Robert"
- the identifier of the Login Plugin is equal to "LoginPluginIdentifier"
- the specific token is equal to "UserAccountObject"

The resulting storage key would be the hash value of the following String word:

$$\textit{Robert} \circ \textit{LoginPluginIdentifier} \circ \textit{UserAccountObject}$$

The uniqueness of the user name makes sure that the storage key is also unique within the system.

The payload of LoginItem contains following information:

- user name
- email address

- hash value of the user password
- node identifier
- NodeHandle object

The NodeHandle object holds the identifier and the current IP-address of the node. It can be used for routing messages from node to node. An alternative way to route messages from Pastry node to Pastry node is to use the node identifier. In LifeSocial.KOM the NodeHandle object is used to route a message to the destination node.

When a new user account is created, a LoginItem object is built. The user name, hash value of the user password and the email address provided during registering are stored in the object's payload. A NodeHandle which detects the current location of the node in the network is also appended to the payload.

Before a user account is created, the Login plugin checks whether the provided user name and email address are not used by other users. If it is not the case, then the LoginItem is stored and the user becomes automatically logged into the system.

LoginItem has an important role in the system. It holds the information required for communication between users. Given a user name, the storage key of the LoginItem asociated with the user account can be generated as it was described above. Using the generated storage key the NodeHandle of the recipient can be retrieved and messages can be routed to this user.

The LoginItem object is expected to stay permanently available in the shared storage. A loss of such data object would cause the loss of all account information associated with the user name held in the LoginItem. Each time a user logs into the system, the associated LoginItem is modified by updating the NodeHandle which holds the current address information about the Pastry node the application runs on.

During registration process a simple user profile is automatically created. The profile contains the user name, email address, country of living and city of living. Login plugin delegates the creation of the profile to the Profile plugin.

Figure 5.3 shows the structure of a LoginItem object. The structure is similar to the structure of the data object which other plugins operate on. All of them extend the SharedItem class to make the objects storable.

LoginItem is used by Message Dispatcher to send user-to-user messages. It is an important object which enables to find users in the network. It can be viewed as a reporting point where all users register their contact information each time they are present in the network. Since the LoginItem objects are stored distributed in the network, no single node in the network is overloaded with queries.

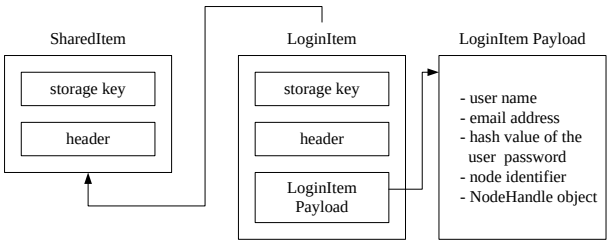


Figure 5.3: LoginItem Structure

5.3.2 Profile Plugin

Profile plugin is responsible for creation and administration of user's profile data. It is also responsible for finding and retrieving the profile data of other users. A basic account is created immediately after the registration of the new user account with help of Login plugin is accomplished. This fact guarantees that every registered user owns a profile with some basic personal information.

The functionality provided by Profile plugin is described as follows:

- create user profile
- retrieve a profile of a user given a user name
- edit profile fields
- store profile (only for own profile possible)

The data objects Profile plugin operates on are represented by the class UserProfile. Like all data objects the plugins work with, UserProfile extends the SharedItem class and hence is a storable object. The payload of UserProfile has the following structure:

- profile image
- profile fields' names with values

Figure 5.4 demonstrates the simplified structure of a UserProfile object.

Image is a profile image which represents the owner of the profile. Profile fields are names of the information the field is associated with. For example, the field name can be "Gender" and the possible values can be "Male" or "Female". Or the field name is "Country" and the value associated with this field is "Germany". Profile field names and the corresponding values are made available in a hash table where the field name is a key

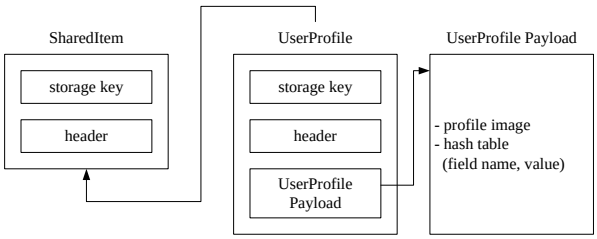


Figure 5.4: UserProfile Structure

of an entry. The number of profile fields can be extended by adding new field names and values. The value of a profile field can be empty.

Directly after the Profile plugin is initialized the profile of the currently logged in user is retrieved from the shared network memory and made locally available. Since the owner of a profile is the only one who can edit the profile contents, the version of the profile which is held locally by the plugin remains valid. If the user profile is changed, the profile is stored in the network and updated locally to have an actual version of the data.

A profile of a user can be loaded by the Profile plugin and made available for the user interface. The user interface instructs the Profile plugin to load a profile for a certain user given his/her user name. Profile plugin checks whether the needed data object is available in the Information Cache and if not initiates loading of the data object from the network memory using Storage Dispatcher. During the loading process, the Profile plugin makes the current status of loading and the data loaded so far available for the user interface. The user interface can check whether the requested profile is loaded and get the data from the plugin.

5.3.3 Search Plugin

Search plugin is responsible for searching of users. Given a collection of criteria, Search plugin makes a list of user names available which are associated with any of the criteria provided in the collection. In order to understand how the Search plugin works, the concept of creating and distributing the data objects used by Search plugin has to be considered.

Every user can be associated with an arbitrary number of criteria which can be used for searching this user. Each criteria is associated with one or multiple user names. The name of a criteria serves as a basis for building storage keys for the data object which holds the list of all user names associated with the certain criteria. A possible criteria can be e.g. the name of the city which is listed in the profile of the user. Assume Berlin

is listed as a home town in the profile of user Alice. This means that this user name is added to the list of other user names in the data object whose key is built using Berlin as a basis. Figure 5.5 demonstrates the structure of a data object which holds all user names of users living in Berlin. The name of the class which provides this structure is SharedItemUserList. It holds a list of user names associated with certain criteria.

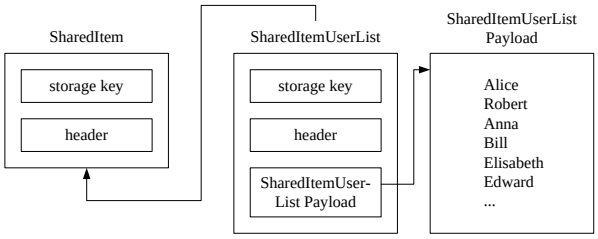


Figure 5.5: SharedItemUserList Structure

The criteria value is embedded into the storage key. The storage key of such object is built according to a predefined pattern. It consists of multiple parts but the only variable part is the value of the criteria. In our example it would be the name of the city Berlin which user names are associated with. Here is an example how such storage key is built:

storage key = hash(Search plugin identifier ∘ Berlin ∘ Search plugin string token)

The "Search plugin identifier" and "Search plugin token" are constant values stored in the system properties. Any criteria is formated in a common way so that independent of the format the criteria is provided to the system, the keys are built in a certain pre-defined way.

In order to execute a search, a collection of storage keys built using the search criteria must be provided. The storage keys are built using the criteria as a key basis. The results are lists of user names. All user names are merged into a single alphabetically sorted list and made available for the requester.

Plugins can register users for certain criteria using a special software component named GroupItemUpdater. The interface provided by this component is defined as follows:

- addUserToList()
- removeUserFromList()

Given a criteria GroupItemUpdater adds or removes the user name of the current user to/from the list of users associated with the given criteria. If the given criteria has not been seen before, a new data object with an appropriate storage key is created.

Using this component, plugins can register users for certain criteria. For example, Profile plugin can register a user for criteria represented by the name of country where the user lives in. GroupItemUpdater takes care about generating the appropriate storage keys, retrieving of data objects from the network and updating the containing lists of user names.

It is possible to implement a range search using the presented concept. The whole range has to be divided into separate spans. With each of these spans a list of user names can be associated. With overlapping spans a better granularity of the search can be achieved. We consider an example which should make it clearly how to organize a range search using data structures based on Distributed Hash Tables technology.

Assume, we want to be able to search for users whose age lies in a certain range. We divide the entire range of all possible ages into spans e.g.

- from 1 to 10
- from 10 to 17
- from 15 to 20
- from 18 to 23
- from 23 to 27
- from 25 to 30
- from 27 to 33
- and so forth

Now assume we would like to find all users who is older than 24 and younger than 28. Knowing the spans of the age range we retrieve two groups of users: from 23 to 27 and from 25 to 30. The results are two lists of user names one for each span. Now by building an intersection of the lists we get a new set of user names. The new set contains the user names of all users who age is between 25 and 27. This satisfies the search criteria. By building a union of the both sets we could get a set of users who is older than 22 and younger than 30.

Assigning users to different ranges makes it possible to determine to what group they belong. The granularity of the range search depends on the number of spans the user has to be registered for. The larger the predefined spans for the range search are, the easier it is to realize the range search in a DHT based storage system.

The Search plugin provides following functionality:

- search for users

  Given a set of criteria it retrieves a list of user names which are associated with some a subset of the provided criteria.

- get available results

  The results of a search available so far can be retrieved at any point of time after the search is started.

- check the completion status of the search process

  The completion status of the search can be prompted.

The Search plugin allows to build associations between users and criteria which characterize the user within the system.

5.3.4 Messaging Plugin

Messaging plugin is responsible for exchanging text messages between users. Every user can send a text message to any other user in the system if the user name of the recipient is given.

Messaging plugin administrates two collections of messages: 'inbox messages' and 'sent messages'. All messages sent by the user are automatically stored in the collection 'sent messages'. All messages which were received by the user are stored in the collection 'inbox messages'. Both collections are permanently stored on the hard disk of the local computer. When the system is started, the 'inbox messages' and 'sent messages' are loaded into the system.

Following functionality is provided by Messaging plugin:

- send a new message

  In order to send a new message to a user, the user name of the recipient and the text of the message must be provided. The plugin builds automatically a message object and initiates the routing of the message to the recipient.

- check for undelivered messages

  Undelivered messages are stored in the network. Messaging plugin provides a method which retrieves the undelivered messages from the network and places then into the 'inbox messages' collection.

- retrieve the list of 'inbox messages'

  Messages, stored in the 'inbox messages' collection, can be retrieved. The list of unique message keys is the result of this function.

- retrieve the list of 'sent messages'

  Messages, stored in the 'sent messages' collection, can be retrieved. The list of unique message keys is the result of this function.

- retrieve a single message from the 'inbox messages' or 'sent messages' collections

  Given a key (unique storage key of the message) a complete single message can be retrieved from 'inbox messages' or 'sent messages' collections.

- information about available unseen messages

  With help of this function it is possible to get information whether new unseen messages are available. All new messages are marked as 'unseen' until they are retrieved.

Messaging plugin provides the basic user-to-user communication. The service provided by this plugin is similar to the service, provided by an email server. However, the current implementation of messages does not allow a message, the Messaging plugin operates on, to contain an attachment. It is only possible now to send text messages within the Messaging plugin. Due to flexibility of the distributed linked list data structure used in LifeSocial.KOM, it is possible later to include pointers into text messages which will allow to build a storage key which points at the corresponding attachment.

5.3.5 Friends Plugin

Every user owns a list of friends. This list contains user names of the users which are registered as friends of this certain user in scope of the system. Friends plugin is responsible for administration of the user's friends list.

When the plugin is initialized, the list of friends of the currently logged in user is loaded. This list remains available within the plugin until the system terminates. Friends lists for all users are stored in the network and are available for every user. The user can request a list of friends of any other registered user. The Friends plugin retrieves this list from the network and makes it available within the plugin. Mostly, two different friends lists are permanently available within the plugin and can be retrieved by upper architecture layers:

- user's own friends list
- the last loaded friends list for a certain user

Every user can retrieve a list of friends of any other user. In scope of this plugin users can exchange messages in form of requests. Requests are simple messages which hold the type of the request and the text included by the user. Following types of requests are possible:

- friendship request

  By sending a message of this type the user invites the recipient to become a friend in scope of the system.

- friendship accepted

  Since a friendship relation in LifeSocial.KOM is always symmetric, both users have to agree in order to create this relation. A friendship request sent by a user can be accepted or denied by the other party. By sending a message of type 'friendship accepted' a user accepts the friendship request sent before. In this case user name of the opponent is appended to the friends list on both sites.

- friendship declined

  This type of message indicates that the receiver of the previously sent friendship request denies the invitation. In this case the friends lists of both users remain unchanged.

- friendship dismiss request

  This type of message indicates that one of the users which is currently listed as a friend has removed the receiver of this messages from his/her friends list. The user name of the sender is automatically removed from the friends list of the receiver. On this way the symmetry of the friendship relation is assured.

The friends lists are modified according to the type of the received request. The user makes a decision about friendship relation to another user which is then informed about this decision with help of a typed request. Users have a possibility to include a text message into a request. The text message may explain the reasons for a certain decision or just contain a self-introduction of the user.

Following functions are provided by the Friends plugin:

- check for undelivered requests

  Undelivered requests are loaded from the network with help of this function.

- new requests available

  This function provides information about new requests which have not been handled yet.

- retrieve new requests

  A collection of new not handled requests can be retrieved with help of this plugin function. Some types of the requests are handled automatically and some of them require the user's decision. In all cases the user is informed about all requests and actions, performed as a reaction to the requests.

- send friendship request

  Given a user name of the recipient, a friendship request can be sent to the user.

- accept or deny friendship request

  If a request of type 'friendship request' is available, the user has the possibility to accept or to deny this request. This function sends an appropriate request back to the sender in order to inform him/her about the user's decision. The modifications of the friends lists are executed automatically according to the user's decision.

- dismiss a friend

  If a friendship relation within the system has to be terminated, then an appropriate request is sent to the opponent.

- load friends list for a certain user

  A user can load a friends list of any other user. With help of this function the loading of a certain list can be initiated. The completion status of the loading process can be prompted.

- get currently loaded friends list

  This function is used to retrieve the currently available friends list which is available within the plugin. The corresponding user name of the list's owner can be also prompted.

- get my friends list

  Using this function the own friends list can be retrieved. This friends list is permanently available within the plugin and has not to be loaded from the network.

Friends lists allow users to navigate through the social network built within the online community. Using the user names, which are contained in the friends lists, different actions can be performed e.g. usage the functions of mandatory or other well known plugins.

5.3.6 Live Chat Plugin

The main purpose of the Live Chat plugin is to provide an instant messaging service for the users. A direct communication between users, which are currently online, can be established with help of Live Chat plugin. The messages, exchanged between users, are text messages. Unlike Messaging plugin the messages sent with Live Chat plugin are delivered directly to the recipient and are not stored in the network. A set of messages exchanged with a certain user is stored in a 'chat history'. The number of messages included in such a set is restricted. The most recent messages build this set. Data objects of the chat history are stored in the network and hence are permanently available for the users. Every user can get access only to his/her own chat history. Each time the Live Chat plugin is initialized, the appropriate chat history data object is loaded from the network and is made available within the plugin.

Following functions are provided by Live Chat plugin:

- send chat message

  Given a user name a new text message is created and routed to the recipient.

- check if new chat messages are available

  The user interface can periodically check whether new messages are available and retrieve them.

- retrieve new chat messages

  This function retrieves a collection of all new chat messages ariived recently.

- get chat history

  As it was already mentioned, the chat history is automatically loaded when the plugins is initialized. The history is updated every time a new message is sent or received.

When the plugin terminates the data object representing the chat history is stored in the network so that it can be retrieved next time the user logs into the system. This data object is an instance of the ChatHistory class. The ChatHistory class is a storable object which holds chat messages. The storage key of a ChatHistory object is generated using the user

name of the owner of this chat history. All messages are arranged into sets using the user name of the chat partner the messages were exchanged with. ChatHistory holds a hash table where the key of each entry is a user name and the value is a set of messages, sorted by arrival time and date.

5.3.7 Photos Plugin

Photo plugin is responsible for administration of albums and photos which users post into the system for access by other user. Every user owns an arbitrary amount of albums (zero or more). Each albums can contain zero or more photos. Every photo is assigned to only one album. Photo plugin allows to load and view photos of other users. The data is organized as follows:

- an 'AlbumKeysHolder' data object holds the storage keys of all albums for a single user
- an album, represented by 'AlbumItem' data object, contains references (storage keys) of photos which belong to this album along with some album information
- single photo represented by 'PhotoItem' data object holds an image and some information about the image

Figure 5.6 demonstrates the way data is organized by Photo plugin:

At any point of time Photo plugin holds information about albums for one single user. In order to view photos of a user, an appropriate AlbumKeysHolder with the storage keys of all albums must be loaded. When the AlbumKeysHolder arrives, the Photo plugin can extract all storage keys of all user albums from it. Using these storage keys, all albums are loaded from the network. The albums are represented by instances of AlbumItem class and contain a list of storage keys of the photos assigned to these albums. The AlbumKeysHolder and AlbumItem object contain only lists of storage keys and hence do not consume much bandwidth while being loaded. Having all AlbumItems of a user we can see the number of photos which are assigned to any of the albums. Photo plugin can load a single chosen PhotoItem object from the network. PhotoItem objects contain the images. In our implementation at any point of time only one image is available within the Photo plugin. Having only one photo available within the plugin is effective in respect to memory consumption. Besides, it is assumed that only one photo is viewed by the user at a time. If another photo should be made available, an appropriate storage key is used for loading this image from the network.

Photo plugin restricts the maximal size of the images. All images loaded into the system by users are automatically resized and fit in a certain size. The image format

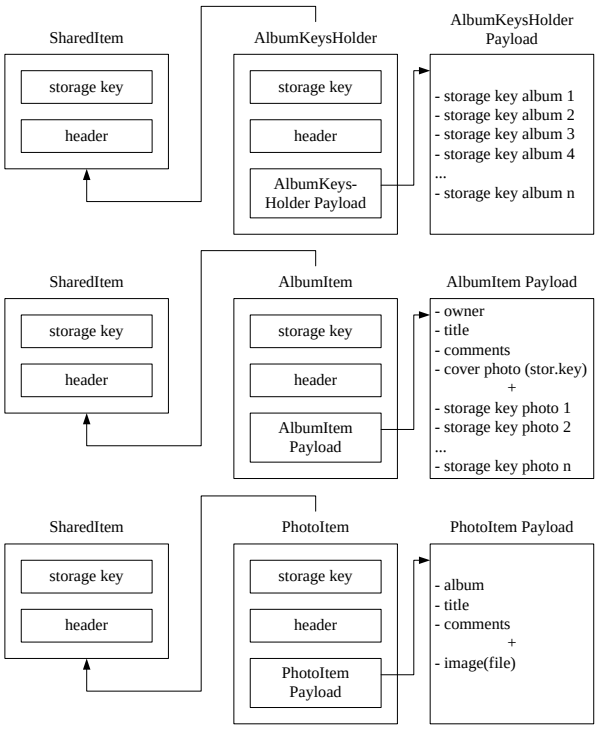


Figure 5.6: Photo Plugin Data Structure

which is currently used by the Photo plugin is JPEG. The number of formats can be easily extended. This will be done in future work.

In order to make it more clearly how Photo plugin works, we consider the following example:

- assume that user Alice wants to see photos posted by the user Robert
- Photo plugin is instructed to load all albums of the user Robert
- Photo plugin builds a storage key using the user name of the user Robert

  $$\textit{storage key} = \textit{hash}(\textit{Robert} \circ \textit{Photo Plugin identifier} \circ \textit{String token})$$

  where String token is a known unique system token and the Photo plugin identifier is the unique identifier of the Photo plugin.
- using the built key an AlbumKeysHolder object is requested. It contains all storage keys of albums which belong to Robert

- when the AlbumKeysHolder object arrives, the list of storage key of albums is extracted and loading of all albums is initiated using these storage keys
- a collection of AlbumItems is loaded after some time and is made available within the plugin
- now Alice can choose any storage key within any album and load the corresponding PhotoItem object which contains the image

Even if the number of albums and photos posted by Robert is considerable, this procedure allows to keep the bandwidth and memory consumption very low.

In order to load all albums represented by AlbumItem objects, Photo plugin uses a special class which is responsible for loading all albums. The name of this class is AlbumsLoader. AlbumsLoader gets a collection of storage keys and loads all albums associated with these keys.

Generally, Photo plugin offers following functionality:

- create new photo album

  Given a title and comments related to the album, a new album can be created.

- add a photo to an album

  Given an album, an image, a photo title and a comment, a new photo can be added to an existing album. The album is indicated by the storage key.

- edit album information

  Given an album, the title and comment of the album can be edited. The album is indicated by the storage key.

- edit photo information

  The information about the currently loaded photo can be edited. The title and the comments can be changed.

- remove an album

  Given an album, the album and all photos which it contains can be removed. The album is indicated by the storage key.

- remove a photo

  Given an album and a photo which is assigned to the album, the photo can be removed. Both the album and the photo are indicated by corresponding storage keys.

- load albums of a user

  Given a user name, the list of the user's albums can be loaded. The loading procedure is executed as it was described above in the example. After this function is executed, a list of albums is available within the plugin. This list is available until the list of albums of another user is requested.

- load a single photo

  Given a storage key of a photo, the photo can be loaded from the network. The storage key can be acquired using one of the available albums.

- the albums and the photo available within the plugin can be retrieved at any moment

  If the process of loading is not accomplished yet, the status of the loading process of the albums or of the requested photo can be prompted.

It is important to mention that users can only alter their own data objects (albums or photos). Photo plugin does not operate on messages. Users communicate only over the shared network storage.

5.3.8 Groups Plugin

Group is responsible for administration of user groups. In this section we extensively describe the structure and design of the Group plugin in order to demonstrate the easiness with which a new plugin can be developed and included into the system.

First we will describe what functions the Group plugin should provide. They can be described as follows:

- create a new group

  A new group should be created given a name of the group.

- join group

  A user should be able to join an existing group.

- send message to all members of a group

  Users should be able to exchange messages within the group.

- load a list of groups for a certain user

  Users should be able to load a list of group of a defined user.

- load all members of a single group

  Given a group name, the list of group members should be loaded.

- get group messages

  A list of messages should be available. Users should be able to read single messages.

- undelivered messages should be stored in the network

We make an assumption that the group names are unique for every existing group within the LifeSocial.KOM implementation.

The functionality of the Group plugin appears to be not very simple. The plugin should use both the shared network storage to store groups and lists of user groups and direct communication with messaging and local message administration. The existing framework makes the development of a such plugin to an issue which can be processed within a short time. The development of the Group plugin alone with the corresponding text-based user interface was accomplished within four working days.

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

1. a list of group names which is associated with a user
2. a list of user names which are members of a group
3. a group message
4. a list of group messages where sent and received messages are stored

For holding a list of group names and the owner of this list we implement a new class by extending the class SharedItem. The payload of the new object contains a list of Strings and the name of the list owner. The structure of the new GroupNamesHolder class is shown in Figure 5.7:

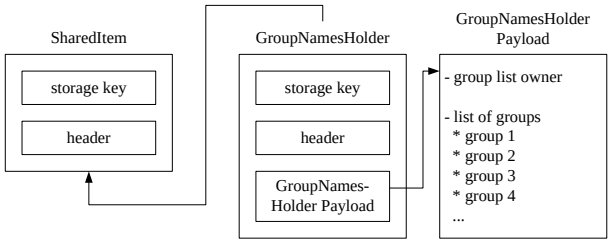


Figure 5.7: GroupNamesHolder Structure

In the description of Search plugin we have already described the class SharedItem-UserList which is used to keep a list of users associated with a certain criteria. We use

this class to hold the user names of a single group. The group name is used as a basis of the object storage keys.

The group message is implemented by extending of a superclass for all messages, BasicMessage. This class is a subclass of SharedItem and implements some additional functionality which a message in FreePastry must provide. Implementing a new message type means to implement a new message payload. The structure of the group message represented by the GroupMsg class is shown in Figure 5.8.

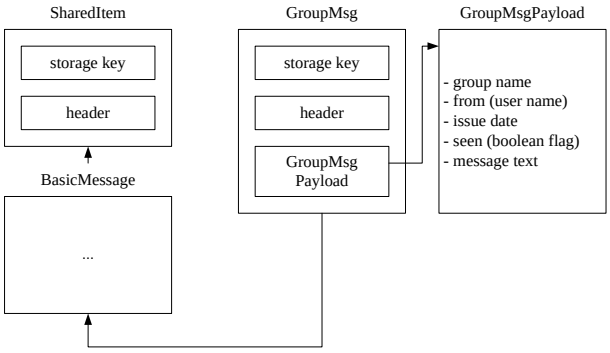


Figure 5.8: GroupMsg Structure

In the Messaging plugin, a special class which holds storage keys of all inbox or all sent messages is used. We use this class to store the storage keys of all sent or received messages within the plugin. The class name is MsgListItem.

At this point we have all classes which are used in the Group plugins implementation. Now a new class named GroupsPlugin is created. This class implements the Plugin interface, the common interface for all LifeSocial.KOM plugins. The functionality of the plugin is implemented following:

- create a new group

  In order to create a new group, a storage key with the name of the new group is created and the object associated with this storage key is requested. The Storage Dispatcher indicates whether the data object exists or not by retrieving the objects or by placing the 'null' value into the Information Cache. If no object with this key exists, a new instance of a SharedItemUserList is created. The user name of the creator is included into the payload of the object and stored using the Storage Dispatcher. A new group is created and the creator is the first group member.

- join group

  In order to join a group, a storage key of the group is build using the group name. Then the SharedItemUserList data object associated with this storage key is retrieved. The payload of the object is extended with the user name of the current user. Then the SharedItemUserList is stored back in the network using Storage Dispatcher.

- send/receive group message (to all group members)

  In order to send a message to all group members, a list of user names is needed. If the appropriate group is already loaded, then the list of user names is available. Using Message Dispatcher and the list of user name a GroupMsg with appropriate content is sent to all users which are registered as members of the given group.

  The messages sent by other users arrive in the Information Cache. A class named PluginMsgsCacheChecker checks for new plugin messages for a certain plugin and then calls a method to notify that new messages are available. This class is implemented and can be used in any plugin which uses message communication. The arrived messages are automatically added to the local MsgListItem object. Both the MsgListItem which holds a list of storage keys of the messages and the messages objects are stored locally. The local storage of the object is processed with help of the LocalStorage class which is responsible for storing and loading data objects to/from local storage capacities like hard disk. The description of LocalStorage is given later in this section.

- load a list of groups for a certain user

  Using the user name of the user, a storage key for the corresponding GroupNamesHolder is built. With help of Storage Dispatcher the data object is loaded and is made available within the plugin. The list can be retrieved using a getter method. The owner of the list is also stored within the plugin and can be prompted using a getter method.

- load all members of a single group

  Using the name of the given group a storage key is built. With help of Storage Dispatcher the data object is loaded and is made available within the plugin. The data object can be retrieved using a getter method.

- get group messages

  A getter method returns the MsgListItem which is available within the plugin after the plugin is inittialized. The existing MsgListItem object is loaded with help of

LocalStorage class from the local hard disk or a new object is created if no MsgListItem is locally stored.

- undelivered messages should be stored in the network

  Message Dispatcher provided the method where given a plugin identifier the undelivered plugin messaged are loaded into the Information Cache.

As it was mentioned earlier in this section, LocalStorage is a component which is responsible for storing and loading data objects to/from local storage capacity. The functions provided by LocalStorage are defined as follows:

- storeItem(String name, String pluginId, SharedItem item)

  where 'name' is the storage key of the data object, 'pluginId' is the plugin identifier of the plugin the data object belongs to and the 'item' is a plugin specific data object. This method stores the data object on the local hard disk.

- loadItem(String name, String pluginId)

  where the 'name' is the storage key of the data object and the 'pluginId' is the plugin identifier of the plugin the data object belongs to. The method returns a SharedItem object if the object with this name is locally stored.

- deleteItem(String name, String pluginId)

  where 'name' is the storage key of the data object, 'pluginId' is the plugin identifier of the plugin the data object belongs to. This method deletes the data object, stored with the defined name and plugin identifier.

Since SharedItem is the superclass of all plugin specific data objects and messages, the developers are enabled to store any of these objects locally with help of LocalStorage.

The implemented framework provides the necessary functionality which makes development of new plugins very easy. A developer just have to extend the existing classes to create new plugin specific data objects. The components like the Information Cache, the Storage Dispatcher and the Messaging Dispatcher provide the functionality which enable the uncomplicated usage of shared network storage and message based communication between applications. The only thing developers of a plugin have to care about are the procedures where the data objects or messages are processed, a clear programming interface for a graphical user interface and a graphical user interface component, which operates with the plugin data objects and messages. After the new plugin for LifeSocial.KOM is developed is has to be placed to a directory where LifeSocial.KOM can detect it and include it into the system.

5.4 User Interface

In scope of this thesis a text-based user interface for the implemented Peer-to-Peer based online community prototype was developed. Graphical user interface is not a subject of this diploma thesis. An appropriate extendable graphic user interface component is the subject of future work of the institute. In this section a short overview over the implemented user interface is given.

Commands is the main user interface class. It contains references to all objects which represent single user commands. Every command is implemented in a separate class which implements a common interface. Commands class holds a hash table in which the String representation of a command is used as a key and the command class instance as the corresponding value. Commands is also responsible for reading the inputs of the user and forwarding them to an appropriate command. An example of a command and its arguments is given below:

login username password

The name of the command in this example is 'login', the arguments are 'username' and 'password'. Commands class reads the command and finds the responsible command class instance in the hash table. The arguments are then forwarded to this instance for processing. After the execution of the command is over, the next command can be read by Commands class.

There exists an interface named CommandHandler which is implemented by all commands classes. The CommandHandler has following structure:

- getCommand()

  This method returns the String representation of the command (name of the command).

- executeCommand(arguments)

  This method is called by the Commands class with all arguments which were provided by the user.

- printHelp()

  This command is responsible to output the command manual text.

AbstractCommandHandler is an abstract class which implements the CommandHandler interface and all methods except the method 'executeCommand(arguments)'. Every command class extends this abstract class and implements the only abstract method - 'executeCommand(arguments)'.

Commands class offers method for registration of single command class implementations. All concrete commands implementations are registered in the Commands class and can be used by the user.

The described framework for creating a text-based user interface is easily extendable. New commands classes have to provide a name of the new command and the action which is processed when the command is executed.

Chapter 6

Architecture Evaluation

In this section, an evaluation of the Peer-to-Peer based architecture presented in this diploma thesis is given. First, the developed Peer-to-Peer based architecture is compared with a typical client/server based architecture concerning robustness, storage behavior, scalability, administration costs and security issues. Then the developed Peer-to-Peer architecture will be evaluated concerning extendability and event propagation.

6.1 Robustness

Robustness is defined in IEEE Standard Computer Dictionary [oEE90] as "a degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions". In respect to the Peer-to-Peer and client/server architectures we consider this property in respect to ability of the system to perform its work if one or a set of the system components fails. In client/server based architecture the clients and the server can be considered as components of a distributed system. If a set of clients fails the server remains unaffected and the performance of the system is not essentially affected. However, if the the performance of the server falls and the server completely fails, the entire system is affected. The entire data stored on the server is not available to the client any more. The latter effect is also called single point of failure. Figure 6.1 demonstrates a screen shot of Facebook platform where a notification about server unavailability is shown.

In the developed Peer-to-Peer based architecture no central entity exists. The data is exchanged directly between peers or stored distributed in the network.The performance of the entire system depends on two factors:

- reliable message routing

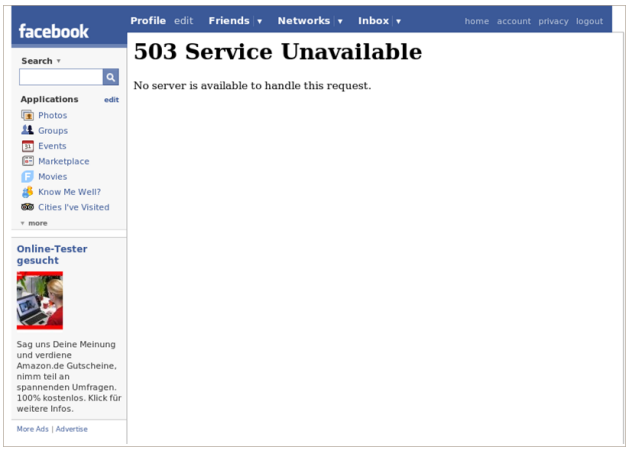


Figure 6.1: Facebook Error

- availability of the saved data objects

If a peer leaves the network, then this fact is registered by the neighbor peers. The messages are not routed over the peer any more and an alternative route is calculated. The data which was stored on this peer is replicated to a new peer which is currently in the network. The larger the number of replicas of a stored data object in the network the smaller is the probability for loss of this data object. The advantage of the developed Peer-to-Peer architecture is that if a set of peers leaves the network or fails, the rest of the network is not affected and the performance of the system built on top of such a network remains almost completely unaffected. Unlike client/server architecture the developed Peer-to-Peer architecture is not sensitive to failures of its participants. The robustness of a system built using the considered architecture model is much higher that the robustness of a equal client/server based system. However, in order to provide high availability of data, PAST [RD01b] hast to be improved.

6.2 Storage Behavior

In a client/server architecture the storage of data is centralized. The more data is stored in the system the more storage capacities must be available on the server site. In the developed Peer-to-Peer based architecture a special storage and replication component built on top of structured overlay network is included. The data objects stored with the storage and replication component are distributed equally over all peers in the network. Similar storage concept is used in the Wuala [GM05] project, a free Peer-to-Peer based storage utility which enables users to store their data using the resources of peers. In order to archive a

better load balancing, the granularity of the data object should be high. Distributed Linked Lists provide a possibility to structure large objects into a set of interrelated smaller objects, each of them is separately stored in the network. Given an equal distribution of data in the network provided by storage and replication component and a high granularity of data objects, an equal load of all peers in the network is achieved. Equal load of storage capacities contributed by the peers makes the consumption of resources of all peers fair. Besides, no central storage capacities has to be provided and administrated. The advantage of the developed Peer-to-Peer based architecture in terms of storage is apparent. Vendors of LifeSocial.KOM may add backup servers for reliable storage until the desired requirements on the distributed storage in Peer-to-Peer overlay networks are fully met.

6.3 Scalability

As defined in [SW05] scalability is a fundamental property of a distributed system which denotes the degree with which a system can satisfy the demand for system's resources like storage capacity, processing power and bandwidth caused by a rapid growth of the number of system's users. In a client/server based architecture scalability is provided by supplying the system with resources like additional storage capacities or processing units according to the growing demand. In order to ensure the system's performance, the system's vendor has to observe the trend of demand for resources and initiate an extension of system's capacities to satisfy this demand.

Unlike client/server based architectures, a Peer-to-Peer based system built on top of a structured DHT-based overlay network is a self-scalable system. Every peer contributes resources to the overall system's resource capacities. The scalability of the developed Peer-to-Peer based architecture depends mainly on the scalability of the overlay network used as a basis of the architecture. In the implementation of LifeSocial.KOM a self-scalable overlay network implementation, FreePastry [DEEH$^+$02], is used. Besides, a storage and replication utility named PAST built on top of FreePastry is used. The overall scalability of LifeSocial.KOM depends on the scalability of the FreePastry overlay network and the PAST storage utility.

6.4 Extendability

As defined in IEEE Standard Computer Dictionary [oEE90], extendability is the development effort which is required to change a software system or a component of a system in order to extend the functional or storage capacity of the system. The minimal the required effort is, the higher is the estimated extendability of the system. In this thesis we will

consider extendability in regard to the plugin layer and functional layers of the developed Peer-to-Peer based architecture.

In the general architecture overview, we stated that every layer can be extended by adding new modules. The plugin layer contains stand-alone software modules which are supposed to interact with the user interface or with other plugins within the layer. The properties and the structure of the architecture provides a possibility to add new components without that other layers or components are affected by the development.

If we consider the prototype LifeSocial.KOM, implemented on the basis of the considered Peer-to-Peer architecture model, we observe that a new plugin can be integrated into the plugin layer by creating a new plugin class which implements a common interface. The components of the lower layer provide communication and storage functionality for the user plugin. Plugins are not restricted in usage of various data structures. Every plugin is free to create its own storable objects and messages with plugin specific payload or user the implementations which are already available. Plugins can work with texts, graphics or any other data formats. The format of the data is not important for the lower layers since the data in handled just as a stream of bytes which can be transmitted to another plugin or stored under a certain key. Every plugin can provide a public interface and functionality which can be used by other plugins located in the same layer. This property increases the reusability of the modules within the system.

On top of a set of plugins new software modules can be build. This modules can accumulate data from multiple plugins and provide new services based on this data.

In order to demonstrate the high degree of extendability of the architecture, we consider an example how a new plugin can be developed and embedded into the LifeSocial.KOM online community prototype.

Assume a Forum plugin has to be developed. The plugin should offer following functionality:

Users should be able to

- create a new forum thread
- retrieve all forum threads
- read a single contribution in a thread
- contribute a new message to a certain forum thread

First an appropriate data object should be created. In order to store a list of all forum threads in a single data object, we use the SharedItemUserList class which holds a list of Strings. The list of Strings will represent the titles of the forum threads. The storage

key of this object is built using the unique plugin identifier and a Forum plugin String token. Using the titles of the forum threads the storage keys of objects representing single forum threads are built. To implement a data object which represents a single forum thread we can use the class KeyHolderItem. KeyHolderItem extends SharedItem class and implements the payload as a list of storage keys. Every single forum thread object will hold a list of storage keys associated with data objects which represent single contributions from the users. Furthermore, each single forum contribution is stored in a separate data object. We create a new class named ForumItem which extends the SharedItem class. The only thing we have to do is to implement the structure of the payload of this object. The payload contains following fields: author, title, text.

The storage key for every new ForumItem is generated randomly. In consists of the Forum plugin identifier, a random String and a Forum plugin String token.

At this point the creation of data object for the Forum plugin is completed. We have to implement a simple container class and use two available implementations. Now the functionality of the plugin has to be implemented. We implement following methods:

- createNewThread(String title)

  This method creates an new instance of KeyHolderItem class and a storage key based on the given title. The storage key for the SharedItemUserList object which holds all thread titles is built and the object is retrieved with help of Storage Dispatcher. We add the title of the new forum thread to the list of all threads and store both objects in the network using Storage Dispatcher.

- getAllThreads()

  This method generates a storage key for the SharedItemUserList object which holds all titles of forum thread. Then the SharedItemUserList object is retrieved. Using the titles of the threads stored in it, a collection of storage keys is generated and the KeyHolderItems representing single threads are loaded with help of Storage Dispatcher. Using the storage keys listed in the retrieved KeyHolderItem object all contributions for every thread can be retrieved.

- getContributions(StorageKey thread-storage-key)

  Given a storage key of the corresponding KeyHolderItem object, the object is loaded and each single contribution can be retrieved from the network with help of Storage Dispatcher.

- contribute a new message to a certain forum thread

  This method creates a new instance of the ForumItem. Then the KeyHolderItem representing the corresponding forum thread is retrieved. The random storage key of the new created ForumItem object is stored in the list of storage keys within the KeyHolderItem object. Then both objects are stored in the network.

An appropriate graphical user interface should be developed in addition to this plugin.

Now the Forum plugin is ready to use. The development of this plugin did not affect any other components of the system. The plugin can be now registered in the PluginMonitor. After that, it can be used by all users who included the plugin into the system.

The given example shows that the architecture concept allows to extend the system with low effort. The clear defined programming interfaces of all components make it possible to use the services offered by these component and to create new components which can use functionality of other existing modules. High reusability of the architecture components make the development effort minimal.

This fact can be confirmed by a simple estimation of the effort needed for development of a new plugin for the LifeSocial.KOM prototype. The estimation is done using the 'lines of code'(LOC) metric which allows a rough software development effort estimation. For this estimation we take the Group plugin which offers extensive functionality and uses both shared distributed network storage and direct message communication. The Group plugin consists of the plugin class and some data object classes which were developed for the Group plugin. The effort for the development of the user interface is not regarded. The total code lines of the Group plugin class and all related classes is equal to approximately 600 code lines. Some classes, which were implemented during the development of other plugins, were reused. Assume that an average programmer can write 150-170 lines of code in Java per working day. This results in approximately 4 full working days of development for a single programmer.

6.5 Event Propagation

An important property of a Peer-to-Peer based architecture is the concept how data objects migrate from the overlay network layer to upper layers and vice versa. We will refer this property in this thesis as event propagation. An event can be an arriving data object, a message or any other information coming from the overlay network layer. Two different approaches were identified during the design process of the architecture model:

1. events are propagated up to Information Cache layer, where they can be accessed by the interested entities

2. events are propagated directly to the receiver entity. A receiver entity can be a plugin or a user interface component for a plugin

After an examination of the advantages and disadvantages of both approaches the first approach was determined as a appropriate one and was taken over into the design of the architecture model. In order to show why the first approach fits better into a Peer-to-Peer based architecture, we consider the way events are propagated using the first approach. Then we discuss how it would be done using the second approach. Finally, we will state the crucial advantages of the first approach over the second one.

In the first approach all events from the overlay layer are propagated up to the Information Cache layer. Arriving data objects and messages are placed into the Information Cache where they can be accessed by the plugins from the immediate upper layer. All plugin messages transmitted by the Message Dispatcher and data objects retrieved by the Storage Dispatcher are collected in the Information Cache. After a plugin has requested a certain data object, the Information Cache is periodically prompted for the object. The loading status of the object is indicated to the plugin as it was described in Chapter 5. This allows plugins to decide what events has to be processed and what events are may be not relevant any more. The disadvantage of this approach is the CPU overhead caused by the periodical queries sent by a plugin to the Information Cache layer. This disadvantage can be taken into account considering the important advantages of this approach.

The advantages of this approach can be stated as follows:

1. upper layers (plugins and user interface layers) are not forced to react to events propagated from lower layers. This makes the internal plugin data management much easier in comparison to another strategy, where the events from lower layers are forwarded to upper layers

2. upper layers can determine the frequency of their queries for related events (e.g. arrived data objects or messages). Some plugins will have a higher querying frequency than the others depending on the character of services the plugin provides

3. the plugins and user interface components are not triggered by the events happening in the lower architecture layers but by the actions generated by the application user

4. lower layers must not be aware about the components in the upper layers. If a new component is appended, it has no affect on the lower layers which remain permanently constant

The second strategy assumes that the events are propagated up to the plugin or user interface layer. This approach avoids that the Information Cache is frequently queried.

All data objects and messages are forwarded directly to the plugins and then to the user interface components. The advantages of this approach are:

- the arrived data is forwarded to the plugins without delay
- applying the observer design pattern, a registration system for the observers (plugins) can be easily implemented

However, the responsibilities and the development effort for single plugins increase considerably because the plugins have to process the data on their own.

The disadvantages of this approach are:

- the lower architecture layers have to be aware of all components in the upper architecture layer which can be receivers of the data objects and messages provided by the overlay network layer. This makes the coupling of the architecture components less loosely than in the first approach
- plugins and user interface components have to react to all events propagated from the lower layers even if these events not relevant any more. The plugin has to care about administration of the data objects
- if multiple plugins request the same data object from the network, then this data object is loaded twice and two different instances of the same object are held in the memory
- user interface has to react to irrelevant events which is expensive in terms of processing effort. User interface has to determine whether the arrived object is the one which was requested by the user or it is not relevant. A complicated internal administration of data is required

To make the advantages of the used event propagation approach more clearly we consider the following example: Assume a user requests a data object which consists of multiple parts. Every part of the object is stored separately and there exists a main object which holds all storage keys of these parts. In a DHT based overlay network like FreePastry and storage utility like PAST this would mean that all these parts are spread among a set of network nodes which are currently in the network. If such data object is requested the responsible plugin requests all of these parts by initiating the loading of each single part. A subset of these parts may arrive very soon and the rest of them may have a considerable delay. The arrived parts are processed and the user gets the currently available data parts. Not waiting until all parts of the data object arrive the user requests another similar structured data object. The plugin requests all parts which build the last requested data object.

The loading of these parts begins. In the meantime the rest of the previously requested object parts arrive. If all data objects are forwarded directly to the plugin entity then the plugin has to process all arriving data object parts and determine whether they belong to the actually requested data object. This would result in redundant plugin activities which would waste processing and memory resources.

Using the event propagation strategy where all events are propagated only up to the Information Cache layer the plugin can just pick the relevant data objects from the cache and compose the requested data object, which can be then retrieved by components from the user interface layer or another plugin. The data object parts which were delayed and arrive later on will be stored in the cache. The delayed data parts would not be pushed to the upper layers. In fact the upper layers (e.g. plugins) would not get notice of arriving of data which is not needed any more. These delayed data objects are then held in the Information Cache and can be accessed by the plugins on demand.

Using this approach has another important advantage: objects which are used by multiple components of upper layers have to be loaded only once. No redundant copies of data objects are held in the application memory.

Chapter 7

Conclusions

In this section conclusions resulting from the work done in scope of this thesis are provided. In the first part the general results and findings of the thesis will be summarized. In the second part the related future work is presented.

7.1 Conclusions

The primary goal of this thesis was to design a Peer-to-Peer based architecture for online communities and to develop a prototype which offers typical online community functionality. The development of such a system helped to research the potentials and limitations of the Peer-to-Peer technology as an alternative to the traditional client/server based technology. In scope of this thesis following work was accomplished:

- the functionality of four currently popular client/server based online communities was analyzed
- a list of basic requirements for a Peer-to-Peer based online community were declared
- a general Peer-to-Peer framework was developed
- the developed Peer-to-Peer framework was adapted to the requirements of an online community
- based on the developed Peer-to-Peer framework, a Java based online community prototype was developed
- during the development of the Peer-to-Peer based online community prototype important observations were made

The observations made during the work on this thesis can be summarized as follows:

- it is generally possible to implement typical client/server function on the basis of Peer-to-Peer technology. Reliable and fail-safe data storage service has to be provided in order to ensure the correct execution of the functions. The availability of some data object (users' authenticity data) must be guaranteed which is a hard issue is a dynamic Peer-to-Peer network.

- the robustness and scalability of the system implemented on the basis of Peer-to-Peer technology can be considerably increased in comparison to client/server architecture

- flexible distributed data structures are required to organize an efficient Peer-to-Peer based storage service

- the integration of an Information Cache layer into the Peer-to-Peer based architecture helped to achieve a number of positive effects like: loosely coupling of the plugin layer with the rest of the architecture, better event processing control, a possibility to define different validity periods for different data objects, decreasing the network traffic over caching strategy, increasing of efficiency over object sharing in the common cache and reducing the development effort for new plugins

- a set of requirements on the security in a Peer-to-Peer based online community was declared. Security issues like authenticity of users, integrity and privacy of data, access control systems for a Peer-to-Peer based online community platform were not handled extensively. Generally, security is identified as an important issue for Peer-to-Peer based applications

- extendability of a Peer-to-Peer based application was researched and an universal extendable framework was developed. Extendability is considered as one of the most important properties of the future Peer-to-Peer based applications (e.g. online communities)

As a result of this work a universal service-oriented extendable Peer-to-Peer based architecture model and appropriate data structures for distributed data storage in a Peer-to-Peer based structured overlay networks were developed. The components of a Peer-to-Peer based application and the component responsibilities were identified. Using an open source implementation of a structured Peer-to-Peer network helped to identify the requirements for the storage and messaging utilities. Development of appropriate components which meet the stated requirements will be possibly a subject of future work of

the institute. The implemented online community prototype (LifeSocial.KOM) allowed to test the Peer-to-Peer technology in connection with a set of new system requirements. The development of a number of plugins gave the basis for estimating the development effort for new plugins (stand-alone applications).

The developed service-oriented Peer-to-Peer based architecture appears to have potentials to serve as a basis for various distributed applications which are currently implemented on the basis of a client/server architecture. The main advantages of applying the Peer-to-Peer technology instead of client/server technology are: higher scalability, better robustness and low administration costs of such systems.

The developed architecture allows to develop stand-alone applications with low effort. The provided framework makes it possible to reduce the development of a new application (plugin) and implementation of new data objects, messages and procedures which process these data. In many cases the implementations of data objects which are already available within the framework can be reused for implementation of new applications.

As described in [Plu05], a service-oriented architecture consists of independent and autonomous services. A service is a software component with a well-defined interface which provides a set of services. The interaction with services is executed over clearly defined messages, which can be exchanged with the software component. The structure and properties of the developed Peer-to-Peer architecture qualifies the architecture model to be a basis of a service-oriented Peer-to-Peer based architecture where plugins operate like services. The extendability of the architecture allows to append new services on demand and makes the architecture a flexible building block for service-oriented distributed systems.

7.2 Future Work

There are three subjects which appear to be interesting and promising research area in respect to a Peer-to-Peer based service-oriented architecture and applications built using this architecture as basis:

- Security issues in a Peer-to-Peer based service-oriented architecture

  Security in Peer-to-Peer based distributed applications is a challenging research area. A concept which allows to ensure authenticity of user and data objects, data integrity and data access control should be developed. The main challenge of a Peer-to-Peer based distributed application appears to be the fact that the entire system data is stored distributed over the network on different network nodes. The most crucial is the storage of authentication data objects. These objects contain

information that identifies users within the network including the user name and a hash value of user's password. Following security requirements were indicated during the work in scope of this thesis:

There are four types of data objects which can be stored in the network:

1. data objects which can be read and modified only by a single user but certain fields can be read by all other users as well (e.g. authentication data)
2. data objects which can be read and modified only by a single user and can be extended by other users (e.g. collection of storage keys for undelivered application messages)
3. data objects which can be read and modified by all users (e.g. a forum thread) thereby simultaneous access to these data objects has to be controlled and conflicting modifications must be resolved
4. data objects which can be read only by a single user (e.g. a private message).

An implementation of various data access policies must be based on a well defined security concept which allows to define different access permissions for different groups of users. The design of a security concept in a Peer-to-Peer based system is a challenging task which will be addressed in future work of the institute.

- Efficient Storage and Replication component

  In the description of the PAST storage and replication utility in chapter five a list of requirements for the storage and replication component was given. An efficient and fail-safe storage and replication utility is a key component of the service-oriented Peer-to-Peer architecture. Issues like data access control and data consistency has to be solved in scope of this component. An implementation of a storage and replication component that meets these additional requirements is a challenging task and will be addressed in the future work of the institute.

- Research of potentials and limitations of the Peer-to-Peer based architecture in application as a service-oriented architecture

  The potentials and limitations of the developed architecture as a service-oriented architecture have to be researched. A precondition for a successful application of the developed architecture is reliability of the communication within the Peer-to-Peer overlay network and efficient fail-safe storage components, which are involved into the model. Given this two factors it is possible to implement the system plugins as services. Extendability of the architecture and the low development effort for new

services can make the concept to a completive technology to the existing solutions in the area of large scale robust architectures.

Considering the advantages of the presented Peer-to-Peer based Framework for online communities in comparison with the traditional client/server based technology, it is imaginable that the Peer-to-Peer technology will be established as the basis technology for some large-scale web applications in the next years. The potentials of the Peer-to-Peer technology are being researched only for a relatively short time. A number of efficient and enormously successful applications like Skype [AH06] are the results of this research. Peer-to-Peer technology is qualified to bring some essential advantages for the application's vendor in the tough market competition.

Bibliography

[Abe01] Karl Aberer. *P-Grid: A Self-Organizing Access Structure for P2P Information Systems*, volume 2172. Verlag Springer Berlin/Heidelberg, January 2001.

[AH06] Baset S. A. and Schulzrinne H. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *Proceedings INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, April 2006.

[All07] OSGi Alliance. About the OSGi Service Platform, 2007. `http://www.osgi.org/Links/Documents`.

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36, December 2004.

[Cas06] Pete Cashmore. Myspace hits 100 million accounts, August 2006. `http://mashable.com`.

[CDK+03] Miguel Castro, Peter Druschel, AnneMarie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. *SplitStream: High-Bandwidth Multicast in Cooperative Environments*, volume 37. October 2003.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.

[Cha07] Jesse Chan. Facebook valuation, November 2007. `http://www.answers.com`.

[Com96] The Web Information Company. Alexa.com, 1996.

[Con95] Randy Conrads. Classmates social network service, 1995. http://www.classmates.com/.

[Cor05] Answers Corporation. Topics encyclopedia, January 2005. http://www.answers.com.

[DEEH+02] Peter Druschel, Romer Gil Eric Engineer, Y. Charlie Hu, Sitaram Iyer, Andrew Ladd, Alan Mislove, Animesh Nandi, Atul Singh Charlie Reis, and RongMei Zhang. Freepastry, 2002. http://freepastry.rice.edu/FreePastry/.

[Fac04] Facebook. Facebook social networking website, February 2004. http://www.facebook.com.

[GM05] Dominik Grolimund and Luzius Meisser. Kangoo: A Simple, Robust, and Efficient DHT for Large Amounts of Data, February 2005.

[Gnu01] Gnutella. The gnutella protocol specification v0.4, June 2001. http://www.stanford.edu.

[Kim00] Amy Jo Kim. *Community Building on the Web : Secret Strategies for Successful Online Communities*. Peachpit Press, 2000.

[LPC+05] Eng Keong Lua, Marcelo Pias, Jon Crowcroft, Ravi Sharma, and Steven Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *Communications Surveys & Tutorials, IEEE*, 7, 2005.

[Mic01] Sun Microsystems. Jxta (juxtapose), 2001. http://www.jxta.org.

[MM] Petar Maymounkov and David Mazières. A Peer-to-Peer Information System Based on the XOR Metric. *International Workshop on Peer-to-Peer Systems (IPTPS), LNCS*.

[MyS06] MySpace. Myspace social networking website, August 2006. http://www.myspace.com.

[New05] BBC News. News Corporation in $580 Million Internet buy, July 2005. http://news.bbc.co.uk/1/hi/business/4695495.stm.

[oEE90] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, 1990. http://www.sei.cmu.edu/str/indexes/glossary/.

[Plu05] Daryl C. Plummer. Defining 'service' is key to implementing a service-oriented architecture, 2005. http://www.gartner.com.

[RD01a] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale Peer-to-Peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[RD01b] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent Peer-to-Peer storage utility. *ACM SIGOPS Operating Systems Review*, pages 188–201, December 2001.

[Res02] Microsoft Research. SimPastry and VisPastry, March 2002. http://research.microsoft.com/~antr/Pastry/download.htm.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable contentaddressable network. *ACM SIGCOMM*, 2001.

[Sar07] Siamak Sarmady. A Survey on Peer-to-Peer and DHT, 2007.

[Sil06] Silicon.com. Myspace heading for $15 billion market value? Reporting by Reuters, September 2006. http://www.enn.ie/frontpage/news-9820797.html.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnany. Chord: A scalable Peer-to-Peer lookup service for Internet applications. *ACM SIGCOMM*, 2001.

[Stö07] Christian Stöcker. Holtzbrinck schnappt sich studivz. Spiegel Online, January 2007. http://www.spiegel.de/netzwelt/web/0,1518,457536,00.html.

[Sto07] Brad Stone. Facebook. *New York Times*, 2007. http://topics.nytimes.com/top/news/business/companies/facebook_inc/index.html.

[Stu05] StudiVZ. Studivz social networking platform (official name: 'studienverzeichnis'), October 2005. http://www.studivz.net.

[SW05] Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Applications*. Springer-Verlag Berlin/Heidelberg, 2005.

[SXY07] Caihong Sun, Yan Xua, and Xiaoping Yang. A tag-based network evolution mechanism for online communities. *Natural Computation, 2007. ICNC 2007. Third International Conference*, 5:487–491, 2007.

[Wal03] Dan S. Wallach. *Software Security-Theories and Systems*. Springer-Verlag Berlin/Heidelberg, 2003.

[Wua04] Wuala. Free online hard-disk, 2004. `http://wua.la/en/home.html`.

[Xin03] Xing. Xing social software platform (named Open Business Club until November 17th, 2006), August 2003. `http://www.xing.com`.

[Zat06] Zattoo. Peer-to-Peer Internet Protocol Television System ("P2PTV"), 2006. `http://www.zattoo.com`.

[ZJ01] J.D. Zhao, B.Y.and Kubiatowicz and A.D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. *Technical Report UCB//CSD-01-1141, U. C. Berkeley*, 2001.