Ackowledgements

I would like to thank all staff from Dublin Institute of Technology for the training provided over the last 2 years. In particular, I would like to thank my supervisor, Damian Gordon, for his patience, support and motivation provided while doing my thesis.

My special thanks to Brendan Murray who provided guidance in technical side of the project. I also thank all the colleagues and friends who have spent their quality time on filling the survey and providing the feedback.

Finally, I would like to thank my family and friends who gave their encouragement and support.

Abstract

In this era of web 2.0, various types of web applications are in practice. Online Social Networking system being one of the popular web 2.0 applications is highly used for social communications. Instant Messaging system is the commonly used communication and collaboration tool in the work environment. Employees of any organisation are shown to rely on Social Networks for some business decisions and problem solving which creates an interest in integrating both the systems. But for various reasons including security, Social Networking systems are not allowed to be used in corporate network (in work environment). Even when allowed, users of multiple Social Networking Systems have to monitor all of them in different interfaces. The project provides a solution for this problem. It uses a popular enterprise Instant Messaging tool, IBM Sametime, as a common platform for using the inherent Instant Messaging features and provides a common interface for any Social Network. The resultant product presents the users with same look and feel, of the Sametime platform, for all the Social Networks integrated.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The advent of web 2.0 enhanced the creativity, communication and collaboration over the internet. This led to the evolution of many internet based applications including Social Networking sites, blogs and wikis. Instant Messaging system, which is a non-web application, existed prior to web 2.0. A web 2.0 application, Social Networking system, and a non-web application, Instant messaging system, are integrated together in this project. The need for such integration, challenges anticipated to face, the aims and objectives of the project are discussed in this chapter.

1.1 Project Background

As a part of working environment or social environment, internet users are always in a need to have some tool that helps them to communicate and collaborate with their colleagues or friends in real time. IBM(International Business Machines Corporation) Lotus Sametime is an award-winning solution for the same need (IBM Lotus Sametime n.d.). It allows corporate users to maintain contact lists, chat, share files with their contacts and use other collaboration and communication services listed in section 1.4. This software includes a server and a client. The server is hosted by the organisation and employees of the organisation use the client which connects to the server, to support communication and collaboration between users. In order to leverage the services provided by Sametime(IBM Lotus Sametime n.d.), all the contacts should be of the same domain that the server is. With new releases, IBM Sametime extended it's services to support AOL AIM®, ICQ®, Apple® iChat™, Yahoo®Messenger and Google Talk™ public IM networks through IBM Lotus Sametime Gateway software. Social networks on the other side are another way to collaborate with people in different organisations and different walks of life. The reason to use a social networking site varies from casual conversation with friends to sharing views on varied topics to sharing media files and comment on them. While this is a common scenario in today's online social world, users have to continuously monitor their friends across different social websites, maintain different contact lists in different interfaces and manage them. Instead, a desktop tool that integrates these social websites into one interface and that provides a common look and feel to all contact lists or contacts would be an ideal solution for this problem.

1.2 What is the need to integrate Social Networking system and Instant Messaging system?

There have been different kinds of researches and organisational moves regarding social networks and their impact on the work culture. Some of them are discussed in this section. Elizabeth (Rosenthal 1997) studied the team relations with people outside the team and the team's performance at work. Her research focused on the pattern of ties that the team members have and how these ties affect the team performance. Her study has provided an evidence for the fact that social or personal networks play an important role in the team performance and the differences in the social networks explain the variations in team performance.

> *"Globalisation is transforming the competitive environment of small and medium-sized firms. Because these firms are competing with their larger counterparts in an economy where collaboration is increasingly central to organisational effectiveness, one must pay more attention to the social networks that organisations rely on."* (Van Laere & Heene 2003)

Van Laere & Heene think that social networks are one way to achieve collaboration in any organisation and participation in social networks can be influential in providing information in timely fashion. According to Burt (Burt 1992), for small and medium-sized firms with a limited amount of resources and competencies available, social networks offer comparatively strong opportunities for a balanced enterprise development in competitive environment. Hence it is favorable for employees of any organisation to maintain both organisational and social networks. Well said by Mitchell-Wong and others (Mitchell-Wong et al. 2007) that online social world is currently not as integrated as real social world and it is of everyone's interest to make online social world in-line with real social world.

> *"Unlike the physical world where social ecosystems are formed from the integrated and managed relationships between individuals and organisations, the online digital world consists of many independent, isolated and incompatible social networks established by organisations that have overlapping and manually managed relationships."* (Mitchell-Wong et al. 2007)

As mentioned earlier, there are a few organisational moves to integrate social networks. Here are two examples of them. IBM Lotus Sametime is an instant messaging solution for collaboration in organisational networks which later extended it's services to few other public instant messaging systems as listed in section 1.1.

1.3 Project Aims and Objectives

In this online social world, people use a number of social networks. Considering the advantages of participating in social networks in work as discussed in the section 1.2, the project's main idea is to provide a solution to integrate organisational network and social networks in some way. The simple and important aspect of this sort of integration is to provide a way to access both the networks. The only way to access social networks in organisations now is through the individual websites. It is quite hard and confusing to maintain and monitor different interfaces for different social websites. The aim of this project is to provide a single and uniform interface to access required social networks. The simple and efficient way to participate in an instant messaging activity is through a desktop tool. The aim is to build a single and uniform interface

for different social websites along with organisational network using a desktop tool. IBM Lotus Sametime being an award winning desktop solution for instant messaging in any organisation, is a good way to provide a platform for collaboration with social networks. The scope of the project when expressed in the terms of aims and objectives will be as follows:

- Objectives of the project:
  - To integrate Enterprise Messaging and collaboration with Social Networks.
  - To deliver the solution as a desktop tool.
- Aims of the project:
  - To use an instant messaging system for communication in organisational network.
  - To provide uniform interface for compatible social networks.
  - To provide the UI (User Interface) that pretends as the UI of Organisational IM (Instant Messaging) system for SN (Social Network) contact lists.
  - To provide as many services of social networks as possible; this is limited by the scope of API (Application Programming Interfaces) exposed by social networks which is an external constraint.

As mentioned earlier, IBM Lotus Sametime becomes ideal solution for communication and collaboration in an organisation. So the aim-1(as described above) of the project is to use IBM Lotus Sametime client. The organisation maintains an IBM Lotus Sametime server. Each employee of the organisation uses an IBM Lotus Sametime client for communication and collaboration in the organisational network. For simplicity, IBM Lotus Sametime will be referred to as Sametime or ST here onwards in this document. Similarly aim-3 is to use the UI of Sametime for social networking sites as well but provide the functionality as offered by the social networking sites. So the users should be convinced that Sametime is providing the services of social networking as it does for the organisational network which is not the case. Aim-4 is to provide the services offered by the social networking sites individually as well as services provided by Sametime wherever possible. The risks foreseen at this stage are

1. The services that Sametime and social sites support are not the same.
2. Technologies or standards that social sites offer their services in are not the same.
3. Data and data formats supported by Sametime and social sites are not compatible.

Sametime supports several features as described in section 1.4 and all of them are not supported by social sites. Social sites offer several features including blogs and feeds which are not supported by Sametime. So a compromise between those services is achieved in chapter 4. All social networking sites do not provide API for developers and the sites which provide development support, do not provide them in same technology or standard. So a generic design is required, which is independent of the API technology, and is covered in chapter 5.

1.4 IBM Lotus Sametime

To achieve the aims outlined in section 1.3, this project makes use of the existing benefits and features offered by IBM Sametime and builds a social application over it. While change in the

range and scope of services provided by social sites is an ongoing business policy, it is impossible to list out a complete set of services that are offered by all the social networks. As a first attempt, here is the list of services or features offered by Sametime that are planned to be exploited in the project. The features (IBM Sametime Features n.d.) are categorised as follows

- **Real-time collaboration:** User presence, enterprise Instant Messaging and Web conferencing.
- **Mobile Access:** Being able to contact whole team where ever the user is.
- **Instant Messaging Federation:** Connect Lotus Sametime with the leading public instant messaging providers.
- **Telephony:** Telephoning contacts.

The enhancements in each category are explained in IBM website (IBM Sametime Features n.d.). To name a few, they are video messaging, tabbed chat interface, screen capture, and web-conferencing.

> *"With it's open and flexible Eclipse framework, Lotus Sametime 7.5 enables customers and third-party vendors to rapidly build and deploy plugins. Incorporating Web 2.0 technologies will enable customers to create new solutions such as mashups, effectively bringing leading capabilities from consumer focused applications to enterprises."* (IBM Press room 2006)

Sametime's provision to extend it using plugins written in Eclipse is the way planned to be adopted in this project. The plugin is meant to integrate Sametime with social networks. The term 'Integration' in the scope of this project refers to the functionality of the proposed product which allows users to maintain their contacts belonging to different social networks inside IBM Sametime client. The UI of the project resembles the Sametime UI where as the functionality it provides can be of social networks.

1.5 Intellectual Challenge

The inconsistencies between Sametime and social sites throw a big challenge and on the other side, different social sites expose their services in different technologies based on which developers would have to build applications for the respective sites. So building a universal application is a difficult task to achieve. One solution is to propose all the social sites to expose their services in a specific technology so that a single application can be run across all social sites. This is an ideal solution to provide uniformity and integration.

The initiative of Google in this regard is it's project called 'OpenSocial'. OpenSocial is an API provided by google which when supported by all social sites can be an easy way to integrate all social websites. This also means that a single OpenSocial application runs across all social sites and can provide integrated functionality of the social sites. While this has been an interesting step towards integration of social sites, the support from social sites is yet to be provided and published. Social sites like Orkut and Ning have already published the OpenSocial support while many social sites listed at Google's site (Google Code n.d.*b*) have announced to join the OpenSocial initiative. Once this project is accomplished, social sites can be integrated using a single set of API.

Inspite of the integration of social sites using the single set of API provided by Google's OpenSocial, there is still a need to integrate it with organisational network which does not come under social network category and hence not compliant with OpenSocial. OpenSocial requires a container to run the application and the containers are the social sites. It also provides SPI (Service Provider Interface) which allows external websites to connect to the social sites as per the requirement (Google Code n.d.*a*).

As discussed above, OpenSocial provides the platform to integrate social sites and any external website. But the aim of this project is to provide a desktop solution for the integration of organisational and social networks. The other option for building social applications is by using the individual APIs that social sites expose. Again, the technology in which social sites expose their APIs is not same. So it will be challenging to provide a single set of API for the proposed integration in order to leverage different technologies of APIs and push the data into the UI of Sametime.

1.6 Thesis Roadmap

The related research and the development life cycle of the project is discussed and explained in the rest of this document. The research related to the areas of Social networking and Instant messaging in home and work environments is discussed in chapter 2. The design methodology used is discussed in chapter 3 and the stages of the methodology used are covered in the chapters that follow it. The project requirements and functionality are discussed in chapter 4. The project architecture and the design based on the analysis done in chapter 4 are presented and explained in chapter 5. The design model is then implemented using suitable technologies and environment as discussed in chapter 6. The software thus created is tested using a test plan. The plan and test cases executed are presented in chapter 7. Finally the project is evaluated in chapter 8 and some future work is presented in chapter 9.

Chapter 2

Literature Review

2.1 Introduction

The concepts of SN and IM were briefly discussed in the chapter 1. These concepts are discussed in detail, in this chapter, with the help of related research. The history, properties, common representations, the importance, purpose, the attitude and behaviour of the people towards SNs are discussed in detail.

The concepts of IM like it's evolution, it's importance, security concerns and how they are addressed are also discussed in this chapter. Finally, the influence of SN in organisations which forms the basis for need of integrating both the systems, that supports the discussion in section 1.2, is also presented.

2.2 Social Networking

2.2.1 Introduction to Social Networking

The concept of Social Networking is not a new one, which came after the advent of internet, but has already existed by then though in a different form. The fact that human being is called a social animal is also supported by the ability of people to work together in groups creating more value than the sum of values created working alone. At a minimum level, Social Network consists of three or more people communicating and collaborating with each other. Such a network can be in any form of social gathering including schools, temples and churches. While such networks can be formed without internet, the advent of internet has created a backbone for virtual social networks where people make their connections for communication and collaboration even without physically meeting them. Such connections between people are seemless and limitless and hence they create opportunity for virtually larger social network than they can imagine in real life. In today's electronic media, social networking is nothing but a network created by individuals using internet, World Wide Web and web applications for their communication and collaboration in ways which were impossible before internet. According to Weaver and Morrison (Weaver & Morrison 2008), the production model has changed to increase the reachability of web applications created for social networking.

> *"The model has changed from top-down to bottom-up creation of information and interaction, made possible by new Web applications that give power to users. While in the past there was a top-down paradigm of a few large media corporations cre-*

ating content for the consumers to access, the production model has shifted so that individual users now create content that everyone can share."

They believe that there is a determinable structure of the way people directly or indirectly know each other and Social Networking is built on this idea. The concept of Social Networks also believes on notions such as "Six degrees of Separation" which states that every person in this world is knowingly or unknowingly connected to everyone else through a chain of not more than six acquaintances (Churchill & Halverson 2005).

The internet based social networking systems allow it's users new and a variety of ways of communication including, PCs (Personal Computers) or even their mobile phones. All users need to do to initiate their network is to create a profile or their own online page on the system chosen and select available users as their friends who are also displayed on their profile. They are allowed to constantly search for new contacts and make them friends by adding them to their contact lists. They can communicate either privately or publicly. Increase in internet access facilities at home, like availability of high speed reliable internet, facilitates the use of social networking systems.

2.2.2 History of Online Social Networking systems

This section presents the history and emergence of social networking systems. While the emergence of concept of social networking is discussed in the section 2.2.1, this section discusses about the evolution of sophisticated online social networking sites.

According to the research of Samantha Lam(Lam n.d.), early social networking sites included Classmates.com (1995) which focused on connecting with former class mates in schools and colleges, SixDegrees.com(1997-2001) which focused on the indirect relationships based on the popular Six Degrees notion. Then came into market, an other popular social networking website named "MySpace" which reported more visitors than "Google". She also mentioned that "Facebook", a good competitor which grew rapidly in size even in exponential rate, overtook MySpace. Later, websites like "Facebook" started supporting external add-on applications for their platform.

The percentage of internet users who contacted various online groups is shown in in the figure 2.1 .

2.2.3 Sociometry (SNA)

In short, Sociometry is a quantitative study of social relationships.

Research in Sociometry:

This section presents some of the research involved in the field of Sociometry including the representation of Social Networks. In their paper, Jamali and Abolhassani (Jamali & Abolhassani 2006) discussed about some important properties of social networks in general. They explained some of the conventional social network models used for representation of relationships in social networks and also for their statistical analysis. The conventional models of social networks, as discussed in their paper, are graphs and matrices used to represent the relationships and statistical models for analysis. These were not always successful, there were some associated problems. For example, statistical models had degeneracy and scalability problems. Handcock (Handcock 2002) discussed the degeneracy problems in statistical models in his paper, where as Hoff and others (Hoff, Raftery & Handcock 2002) discussed the scalability problems in their

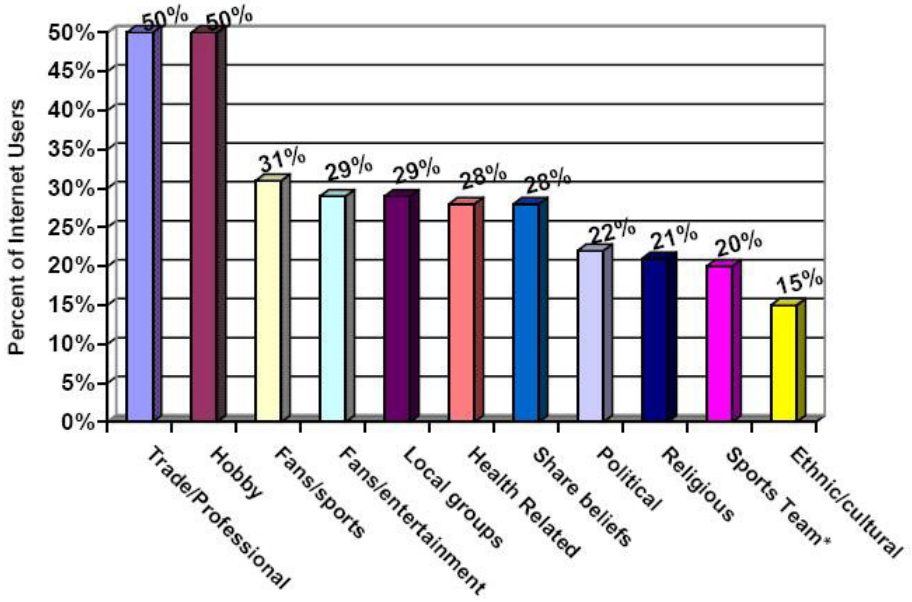


Figure 2.1: Types of Online Communities and percent of internet users by online groups to which they belong (Source: Pew Internet & American Life Project, Jan.-Feb. 2001 (Survey, Internet users, n=1,697. Margin of error is + or – 3% (Project, P.I.A.L 2002))

papers. The graph and matrix models have shown readability problems with large number of nodes and connections. Jamali and Abolhassani (Jamali & Abolhassani 2006) represented social networks using nodes and edges where nodes represent actors and edges joining the nodes represent the relationships between two actors. Other important aspects in social network representation (where importance is based on frequency of these definitions in literature) are outlined in the figure 2.2.

During 1930s, sociometric analysts in US investigated how the feelings of well-being related to the people's social lives. Elizabeth and Christine (Churchill & Halverson 2005)found this research closely associated with the sociogram devised by Jacob Moreno. A Sociogram is a visual diagram which represents the people and the relationships between them using points and lines where a point represents a person and a line joining two points represents the connection between those people. They also stated that Kurt Lewin and Fritz Heider were among other major players in this research. Kurt Lewin promoted mathematical models of group relations and Fritz Heider focused on perceptions of people about their relationships.

Figure 2.3 is an example sociogram which depicts the structure of relationships between nodes (which represent people) or entities A through G. In the figure 2.3, elements of a social network are illustrated as a simple sociogram. The nodes in figure 2.3 are represented by circles and the connections or relationships are represented by arrowed lines. It consists of both unidirectional and bidirectional lines. Node A links two subgroups of linked nodes consisting of B, C and D and E and F respectively. A single node G is also connected to A. Another interesting point is that A connects to E but E is not connected to A. The lines are also called links or arcs or

Name	Definition/Description
Degree (of a node)	Number of ties for an actor
Closeness (path length)	Lengths of paths to other actors
Betweeness	Lying between each other pairs of actors
Clique (hub/clusters)	Actors who have all possible ties among themselves
N-Clique (hub/clusters with diameter N)	Actors are connected to every member of the group at a maximum distance of N
Component (core)	Parts of graph that are connected within but disconnected with other components
Cut Points	Nodes which if removed, the structure becomes divided into un-connected systems
Block	The divisions into which cutpoints divide a graph
Lambda Set	Set of actors who if disconnected, would most greatly disrupt the flow among all of the actors

Figure 2.2: Some properties of a Social Network. (Source: Representation of Social Networks (Lam n.d.))

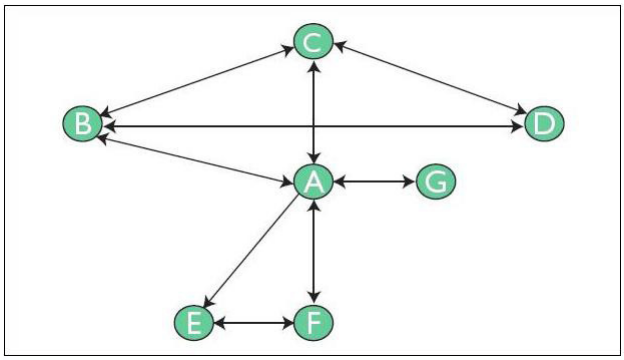


Figure 2.3: Elements of a social network, illustrated in a simple sociogram

edges or ties. The single nodes like G are also called singleton. B, C and D form a subgroup so do the nodes E, A and F. If the relationship is defined as "*sending wedding invitations to*", the links from the network explain the following.

- A sends invitation to B, C, E, F, and G.
- A receives invitations from B, C, F and G.
- E and F invite each other
- B, C and D send and receive invitations among themselves.
- A does not receive an invitation from E.

Example analysis from the figure 2.3 can be that A is serving as potential connection source between the members of two subgroups as it is spanning the boundary between them. From the graph in figure 2.3 , a connectivity matrix can be visualised as in figure 2.4. In the matrix, the cells are binary digits '0'and '1'. A '1' in a cell indicates that there is a connection between the nodes represented by the row and column of that particular cell and in the same way a '0' indicates that there is no connection between the nodes. The directional link is interpreted as follows – the row represents the 'From' node and the column represents the 'To' node. For example, A is connected to E but E is not connected to A and hence row A and column E has 1 and row E and column A has 0. A node is supposed to be connected to itself and hence the leading diagonal in the matrix is 1 (the cells like A-A , B-B are always 1)

	A	B	C	D	E	F	G
A	1	1	1	0	1	1	1
B	1	1	1	1	0	0	0
C	1	1	1	1	0	0	0
D	0	1	1	1	0	0	0
E	0	0	0	0	1	1	0
F	1	0	0	0	1	1	0
G	1	0	0	0	0	0	1

Figure 2.4: Connectivity matrix for entities A through G in Figure 2.3 (source – (Churchill & Halverson 2005))

2.2.3.1 Why do these social Networks matter?

With any possible representation of social networks, the questions that arise are why should a social network be represented in any model, what can be derived from it and will the derivations be useful and so it is very important to understand or analyse what is already represented by a given social network. The availability of data to analyse is directly or indirectly proportional to some factors including the number of people using the social network systems and the amount of their interactions. More availability of such data makes analysts interested in observing and characterising the details about connections including their origin, timing period, the way they are made and how they will be helpful. There are many researches done at the level of interpersonal communications and interests. Wellman and Gulia (Wellman & Gulia 1999) examined the social information flows for both kinds of ties or links, strong and weak. Kraut and others(Cummings, Butler & Kraut 2002) examined how online social relations influenced the psychological health of people. Hampton and Wellman (Hampton & Wellman 2003) examined how these relationships and interactions affected face to face communications and interactions in real life communities, Sunden (Sundén 2003) studied how people represent themselves through constructed online identities, which is also called impression management.

Despite no business model being emerged by then, companies like LinkedIn, MySpace, Orkut, Friendster and Tribe became increasingly interesting in social network concepts and were successful in launching social networking websites. Social Network Analysis was also used

- To analyse the ways of communication & cooperation that people used
- For the identification of knowledge flows like the source of information or knowledge among the people and who do they share this information or knowledge with.

SNA (Social Network Analysis) when applied in the context of a business, the informal communication networks could be revealed and the source or the flow of information within the organisation between the formal procedures and relationships could be identified. Elizabeth and Christine (Churchill & Halverson 2005) observed that several consultancy firms offered SNA based services and promised to improve productivity, efficiency and reduce costs by optimising the information flow. In the context of research, the process of understanding the flow of informal

networks in inter and intra organisations lead to a separate area for study called Organisational Network Analysis (ONA). Cross and others (Cross, Nohria & Parker 2002) have contributed to this area of study.

2.2.3.2 Attitude and Behaviour towards Social Networking Sites

Although there were enormous number of social networkers, their attitude towards Social networking sites was different and the reason to use the them was not the same. Some research is done by OfCom, UK (Ofcom, Office of Communications 2008) in this area. According to Ofcom's qualitative research, social networkers could be differentiated into five groups based on the difference in their attitude to the social networking sites and their behaviour while using them. These groups are described as follows.

- **Alpha Socialisers:** The people who use the social networking sites very often for flirting, meeting new people. Their main motive is being entertained. There were only minority of people of these type at the time of research.
- **Faithfuls:** The people who used the social sites to maintain their old friendships generally from their school or universities. According to the report, there were many people of this kind.
- **Attention Seekers:** The people who used the social sites to seek the attention from people and were so keen on obtaining comments from others about their profiles or photos uploaded or anything published about them. There were only some part of social networkers of this type.
- **Followers:** The people who joined the social sites just because others were doing or to keep up with the changing technologies or to keep up with their peers.
- **Functionals:** The people who joined the social sites for specific purpose. There were only a minority of social networkers of this type according to the report.

While the majority of internet users were part of Social networking, there were people who were not using the the sites for varied reasons. The report (Ofcom, Office of Communications 2008) also classified the non users of social networking based on the reason for not using them as follows:

- **Concerned about safety:** Many people were afraid of using the social sites because they were afraid of publishing their personal details on the internet. They were concerned about the safety of making their details available online although technically they would not need to publish legitimate or correct details.
- **Technically inexperienced:** There were a portion of non users of social sites who were not aware of using the technology and who lack the confidence in using computers and the internet.
- **Intellectual Rejecters:** These type of people would not have any interest in using these sites at all and see using them as just a waste of time.

2.2.3.3 Consequences of using Social Networks

Although social networking was intended to keep up social relationships, personal and work related collaborations, the websites have some negotiable pitfalls such as unwanted consequences of the availability of sensitive personal information to the public. Inspite of the safety measures taken by the websites, there were possibilites where users were confused over the privacy settings and contacting with strangers online. The report (Ofcom, Office of Communications 2008) stated that some users were annoyed by few activities including self-promotion and online bullying. The report indicated that many users were very positive about using the social networking sites and that there were only a few who reported negative experiences. The research suggested several areas of potentially risky behaviour including:

- **Privacy settings:** Some users left their privacy settings as open by default. It stated that 41% of children who were aged between 8 & 17 and 44% of adults aged between 18 & 24 left their profile visible to everyone.
- **Giving out sensitive personal:** Some users willingly gave out their sensitive personal information like photographs to everyone. 25% of registered users posted sensitive personal data like phone number, address and email address on their public profiles.
- **Giving out reputationally damaging information:** Some users posted reputationally damaging information about others on their pages such as provoking photographs, photographs of their teachers or bosses drinking or smoking.
- **Contacting complete strangers online:** Many users of all age groups added complete strangers as their friends and share their personal information. 17% of adults contacted complete strangers and 35% contacted friends of friends

Although the purpose of social networking is to increase their social networks by knowing new people and communicating with them, it can still be a security risk. Inspite of the outlined risks of security many people appeared unconcerned due to various reasons including

- Lack of awareness
- False assumptions that social sites would take care of privacy and security issues.
- Lack of knowledge about managing their privacy settings
- Not so easy availability and understandability of the privacy and safety measures on the internet.
- Perception that social networking security and privacy concerns would not be so serious as internet banking systems.

2.2.3.4 Web 2.0

Social networking websites are also part of the wider Web 2.0 context. OfCom (Ofcom, Office of Communications 2008) defines web 2.0 as

> "*The specific technology that has enabled this growth in the number and popularity of social networking sites is part of a wider online phenomenon, enabling self-expression, communication and user interaction online, known as Web 2.0.*"

Though Web 2.0 is not unique to social networking, it has eased the development of several interactive and collaborative applications including user generated content websites also called UGC (example: YouTube and Picasa), file sharing websites and Massive Multiplayer Online Role Playing Games (MMORPGs. Example: VirtualWorld, Second Life)

2.2.4 Popular Social Networking Softwares

The table 2.1 shows the list of popular websites, when they were founded and the active number of users at 22 August 2007 as researched by Forbes (Forbes 2007)

Table 2.1: Popular social networking sites based on number of active users

Social Network Site	Founded In	Number of Active Users
Windows Live Spaces	December 2004	130 million
MySpace	July 2003	115 million
Orkut	January 2004	60 million
Friendster	2003	47 million
Classmates.com	1995	40 million
Facebook	February 2004	34 million
Flickr	February 2004	11 million

Inline with numerous media reports, the most used social networking websites as reported by OfCom (Ofcom, Office of Communications 2008) were Facebook, MySpace and Bebo. The report also indicates that about 62% of adults who use social networking sites are also a user of Facebook, 50% used MySpace and 33% used Bebo.

2.3 Instant Messaging

2.3.1 Introduction to Instant Messaging:

Instant Messaging is a simple but most liked and used concept where users exchange text messages in real time using a software application. It is often called in short as “IM” or “IMing” (also referred as IM in this report here onwards). Generally IM softwares allow features like maintaining a contact list (or buddy list), sending and receiving messages from contacts in real time and see the online status of the contacts. Instant Messaging softwares were designed for both home and office use. Some high end IM applications also provide many features including file transfer in real time, spell check and call in realtime.

2.3.1.1 Origin and Evolution of Instant Messaging

Most of the internet users use emails(short name for electronic mails) for communication, which was developed by ARPAnet in 1972 (Project, P.I.A.L 2002). Early email systems were only point to point which means that a person could only send an email to only one other person and not more. Later Listservers were invented in 1975 which allow one to many postings. Jenny Preece and others (Preece, Maloney-Krichmar & Abras 2003) in their report mentioned that the first emoticon (short name for emotion-icon which is a short and brief graphical representation of an emotion. Something like ':-)' to indicate an emotion of being happy) was invented in 1979 by Kevin Mackenzie. The usage of such emoticons softened the impact of otherwise dry email text by making it interesting to send and receive. It is also said that “ *A picture is worth 1000*

words". Later in mid 1980s, communication tools like email systems were improved by allowing graphical user interfaces (GUI).

As per the report (Preece, Maloney-Krichmar & Abras 2003) the next in the evolution were online bulletin boards which were designed to work like physical bulletin boards. These boards allow it's users to post any messages and those messages were displayed in various ways. Thus posted messages were threaded to mean that messages related to a topic were associated or linked with each other. Threading messages allowed users to navigate from first message of the thread to later messages and again backwards for example providing 'next' and 'previous' links. Later systems appeared to offer many enhancements including allowing users to search on topics, users who posted messages, date of posting messages, links to email, displaying user profiles and web pages, two dimensional graphical representation of people involved called "avatars". At this stage the normal contact lists turned into online communities based on the scope of the system. The communication tools like email, list servers, bulletin boards, Usenet news are asynchronous communication technologies which means that the users communicate with their communication partners even when both parties do not co-present in time.

The Synchronous communication systems include chat systems, instant messaging and texting systems which means that the communication partners have to be co-present. Generally conversations were rapid and short. Texting systems were like instant messaging on phone lines. The report (Preece, Maloney-Krichmar & Abras 2003) also stated that in 1991 (an year after ARPAnet ceased in existence) the World-Wide-Web (WWW) was developed by Tim Berners Lee and was released by CERN (European Organisation for Nuclear Research). WWW then facilitated widespread usage of web sites. In the area of online communications, WWW eased the development of several online communities and many forms of communication software.

2.3.2 IM at work

Real time communication and collaboration was possible using IM softwares at homes. But designing effective real time collaboration tools for people at workplace was heart of research in computer cooperative work (CSCW) says QhanHeese and others (Quan-Haase, Cothrel & Wellman 2005) The security issues, the effective utilisation of IM software among the employees were key issues in employing IM at work. Nardi and O'Day (Nardi & O'Day 1999) think that CSCW could not be developed solely online.

> *"Successful development and deployment must take into account the social context of use, and must understand the situations in which users switch among different means of communication"* says Nardi and O' Day (Nardi & O'Day 1999)

Many researches (Handel & Herbsleb 2002, Herbsleb et al. 2002, Poe 2001) have shown that in many organisations, employees use IM. IM systems were being used in either way with or without email systems. Speed and ease added to the workplace communication by IM made it so popular in workplace. It was also observed (Nardi, Whittaker & Bradner 2000, Handel & Herbsleb 2002) that IM eliminates the time lost to "telephone tagging" or making fruitless trips to absent coworker's office as IM requires not more than an internet connection along with a small application launched on the computer.

2.3.3 IM and Security

As mentioned in the section 2.3.1.1, IM systems have improved from providing real time text message transfer to providing transfer of files. This feature posed a risk of carrying worms and

other malware from one computer/network to another similar to what emails could do. Thus IM systems provide an access point for backdoor trojan horses. They can also be used by hackers to obtain backdoor access to otherwise safe computers/networks without any open port listening to it and hence bypassing desktop and boundary firewalls as in figure 2.5

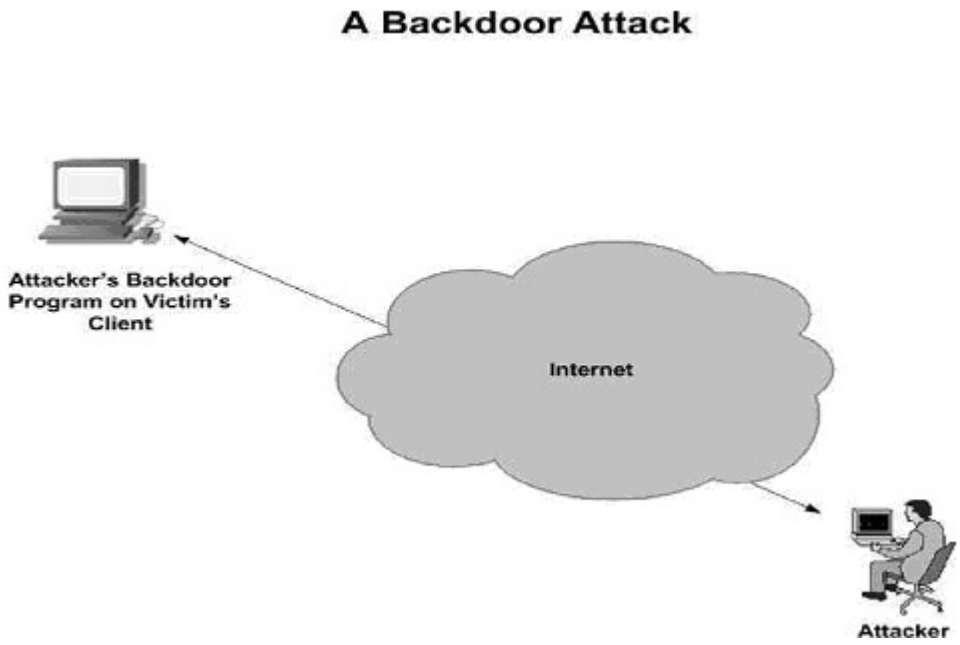


Figure 2.5: A backdoor attack (source: (Rittinghouse & Ransome 2005))

Peer-to-peer file sharing which is another security threat that allows a directory or drive to be shared is also allowed in IM. This feature allows all files on a computer to be shared and thus leads to the spread of virus or malware affected files to other systems. Viruses are transfered to other computers, bypassing the gateway anti-virus software. The text messages being transfered can also be urls for pages that contain malicious code on the internet. Rittinghouse and Ransome (Rittinghouse & Ransome 2005) mentioned that even the use of organisational IM systems like IBM Lotus Sametime and Microsoft Office Live communications, the communications will be safeguarded internally and hence the communications, once enter the public networks, are again exposed to security threats.

However constant improvement in security features were made for major enterprise solutions like IBM Sametime which is used in this project. So whatever choice has been made by a company to employ an IM system, be it popular and free IM or corporate-focused and secure IM, and again deploy suitable security layers such as antivirus software, firewalls and vulnerability management on top of IM solution, it has to be aware of associated security risks.

Rittinghouse and Ransome (Rittinghouse & Ransome 2005) reiterated that architecture and design of the IM should always reflect an effective security strategy and in simple words it's security strategy must employ the sophisticated techniques in establishing the connection with IM service.

2.3.3.1 Using Firewalls

Improvements and steps taken against existing security risks to IM systems always results in new security threats being identified. So out-of-the-box firewall configurations often would not be sufficient for blocking unwanted access to the IM systems. Latest IM systems were designed

keeping firewalls in mind and they employ many techniques to sneak past corporate firewalls for reaching the IM servers as mentioned in the book (Rittinghouse & Ransome 2005).

Some companies block non trustable or non approved IM system usage in the internal network. In order to do that, the IM clients must be prevented from reaching the IM server by adding the servers (names or ip addresses) to the firewall block list for each of the IM services to be blocked. The usage of firewalls is illustrated in figure 2.6.

Figure 2.6 illustrates the usage of perimeter firewalls in organisational networks which again are connected by perimeter network.

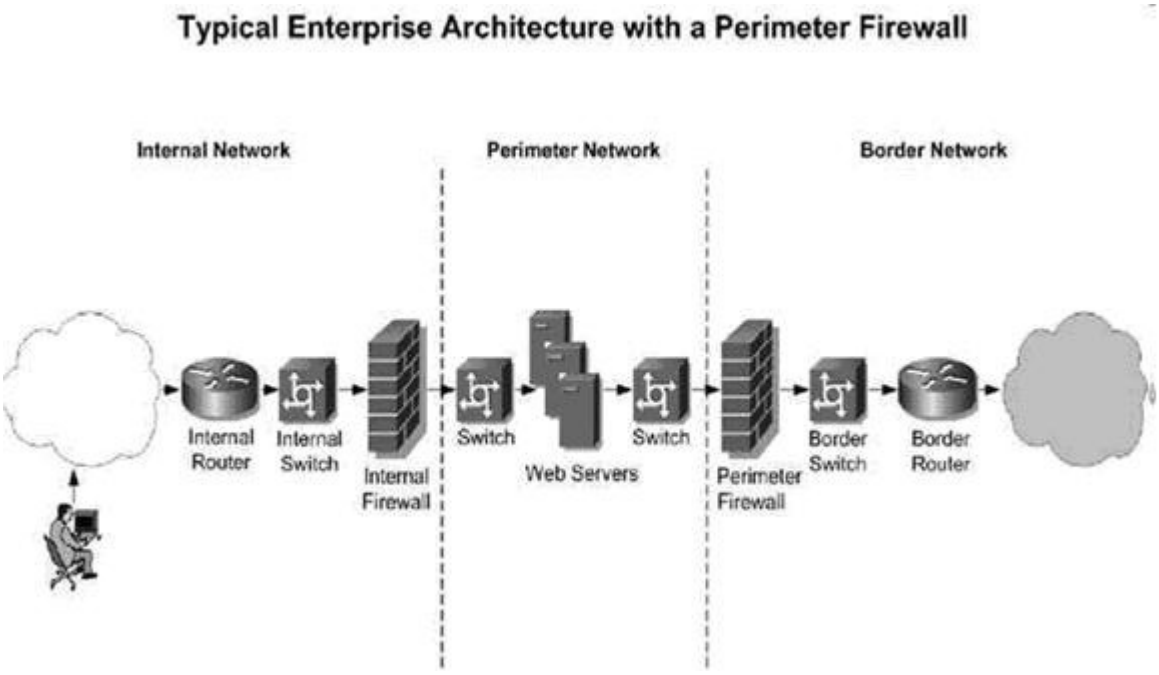


Figure 2.6: An illustration of a typical firewall architecture in an enterprise (Source:(Rittinghouse & Ransome 2005))

Perimeter firewalls are used for blocking non approved or non trusted IMs. Generally organisations configure their perimeter firewalls so that all internet services are blocked other than critical set of services like SMTP email, HTTP Web surfing and DNS. This led IM providers to design their software so that the clients tunnel over the allowed set of internet services and thus they slip past the firewalls.

2.3.3.2 Blocking and Proxying IM

As discussed in the section 2.3.3.1, IM services on the whole could be blocked by firewalls, so that IM client can not be able to connect to the IM service provider or the IM server. But an effective way to block the IM can be allowing the connection to happen, later intercepting the connection and reject the user log-in packet. So from the end user perspective, it appears like a normal failed log-in attempt but not that the connection was unsuccessful. Another way can be, allowing the login to happen but intercepting or rejecting all other message packets and showing a warning to the user that IM usage is not allowed. Rittinghouse and Ransome (Rittinghouse & Ransome 2005) proposed using an application level proxy server for this purpose. The proxy server can be configured based on the security policies determined by IT manager and this would be an added advantage to organisations. The to and fro messages and files can also be scanned for any malicious content or filter URLs using the proxy server.

2.4 Social Networking and Organisational influence

2.4.1 Entrepreneur level influence

Many researches (Borch 1994, Hansen 1995, Larson & Starr 1993, Reynolds 1991, Starr & MacMillan 1990) have shown that business decisions were always embedded by entrepreneurs in social structures. A structural approach was used by Greve and Salaff (Greve & Salaff 2003) to study the way entrepreneurs used their social relations to help launch any business. Entrepreneurs consider any advice and resources they obtain from social relations, as different contacts and resources are always a requirement to establish any business. The structural approach used by Greve and Salaff also implied that entrepreneurs used social relations, to obtain any useful resources, in diverse cultural settings. They obtained and gained access to resources like support, knowledge and distribution channels through their social networks and would interact with other people and organisations, to widen the resource availability and try to sustain their new firms.

Donald McDonald (McDonald 2003) stated that social networks are a way to visualise the interactions in organisational settings. Their descriptive and analytical power made them popular and useful in designing groupware systems.

2.4.2 Other Organisational benefits

Social networks also are used in many systems as a mechanism for recommending sources for collaboration. Visualisation techniques are used to find specific people who served as most common source for collaboration. An example of such systems is a ReferralWeb (Kautz, Selman & Shah 1997). Co-authoring and co-citation relationships were mined to create a social network and the visualisation of that network was used to find a subject expert. Such a social network was also used to answer other queries like how far away each researcher was located and if any one was in between them.

Some studies (Ackerman 1998, George Chin, Myers & Hoyt 2002, Ehrlich & Cash 1999) regarding collaborative behaviour found that any single person's social network influences other people's collaborative behaviour and information seeking. In these studies, social networks showed a broad and complex range of social and organisational interaction and hence demonstrate the importance of social networks in workplace. Expert finding system is a type of recommendation system which is designed to discover a person who can be treated as subject experts in a specific problem domain. These were used to locate expertise in any unfamiliar part of an organisation and they also provide alternatives in the key expert's absence.

2.5 Conclusion

The concepts of SN and IM, their importance in an organisation and the behaviour of people with the SN systems are discussed in this chapter. The very idea of the project, to integrate both the systems, is also supported by the discussion in section 2.4.

Now that the need to integrate them is clear, the life cycle stages of the development of the project are covered in the rest of the document. Development methodology used is explained in chapter 3 which guides other stages.

Chapter 3

Development Methodology

3.1 Introduction

The concepts of SN and IM are discussed in chapter 2. The need to integrate both the systems, based on existing research, is discussed in the sections 1.2 and 2.4. Having supported by the research, the project life cycle started by choosing an appropriate development methodology which is then developed in various stages.

The activities that are involved in a project or a system are defined using a development methodology or a system methodology. The activities of developing and evolving systems, right from the initial feasibility study to when the system is ready and fulfills the requirements laid, are defined by the methodology employed. The continuity in the process of completing a project is supported by use of a particular methodology. According to Keyes(Keyes 2003), in a corporate environment, it is the methodology employed that assures the organisation, that the process of developing and maintaining the system (project) is sustainable and repeatable. The development methodology used for the project is discussed in this chapter.

3.2 Comparison of development methodologies

The following development methodologies are discussed in this section to identify the most appropriate methodology for the project.

- Waterfall model
- Agile Methodology - Extreme Programming
- Rational Unified Process

The benefits, shortcomings and the suitability of these methodologies are discussed in this section.

3.2.1 Waterfall Model

Waterfall model, also called as linear sequential model, is a sequential or systematic approach. It suggests that the stages in software development have to be sequential.

3.2.1.1 What is Waterfall model?

The figure 3.1 illustrates the model. Although this model is generally derided as old-fashioned, when the requirements are well understood in advance, it can do reasonably well because of it's simplicity. The main activities or phases or stages of this model are Analysis, Design, Implementation, Testing and Maintainance as shown in figure 3.1.

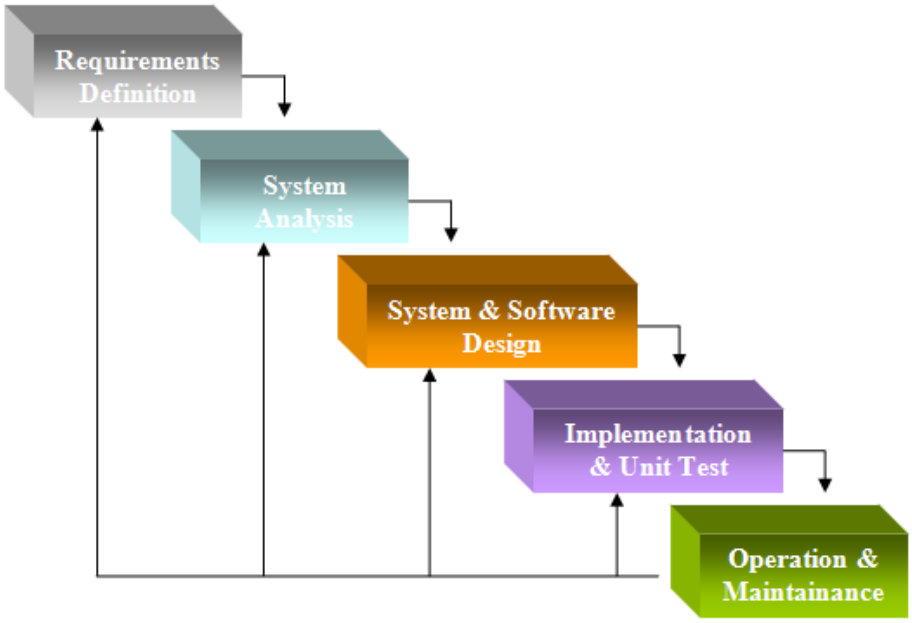


Figure 3.1: Waterfall Model

Bradac (Bradac, Perry & Votta 1993) analysed few real time projects which follow this methodology and found that the each stage blocks the stages next to it and leads to the respective teams to wait in case of dependencies.

3.2.1.2 Benefits

- It is very simple, and so would have less learning curve, and well structured.
- It is a systematic approach organised stepwise which provides much clarity.

3.2.1.3 Shortcomings

- The analysis and specification should be done at the starting stages of the projects and never revisited, which makes it practically difficult for real time projects.
- A prototype of the product is obtained at later stages so this might not be what clients want.
- The strict structure of stages require that future problems have to be anticipated at the early stages itself, which might not be practically possible.

3.2.1.4 Why the project can not employ it?

Eventhough there are no teams to wait for other teams in case of any dependencies, the absolute sequential model can not be employed for the project because of the constantly changing features

(hence requirements) and development support of the products used (Sametime as well as Social Networks). So some kind of incremental or iterative procedures are needed for this project.

3.2.2 Agile methodologies

According to Fowler(Fowler, M 2000), in the field of software engineering, the term 'Agile' refers to a philosophy of software development rather than a simple process or methodology . There are many approaches in agile including "Extreme Programming", "Scrum" and "Crystal".

3.2.2.1 Description

The main focus of agile methodologies according to Abrahamsson and others(Abrahamsson et al. 2002) are listed as below

- **Individuals and Interactions:** The relationship between software developers is more emphasised than the processes and tools. The team spirit, close team relationships and close working environment are enhanced by the agile practices.
- **Working Software:** The working version of software is emphasised more than producing comprehensive documentation. Simple, efficient, technically advanced and reusable software is demanded for producing frequent releases of the software.
- **Customer Collaboration:** The customer, developer relationships are given precedence over contract negotiations. From a business point of view, the sofware is expected to deliver business value from the day one and thus reducing the risks of contract non-fulfillment.
- **Responding to Change:** The software development team along with customer representatives should be informed about any changes or adjustments that might emerge during the development process.

Agile methods are very deliberate and disciplined approach to software development. It brings the whole team together by a set of simple practices and enables the team to know where the project is and allows them to change the practices to ensure the productivity. Agile methods stress on customer satisfaction and are designed so that the product is delivered to the customers as and when they need. They emphasise team work. They encourage the team members to complete the project as early as possible and also allow customers to give any new requirements or alter the requirements even in the later stages of the development. The practices in agile methods are team oriented and hence more suitable for team environments.

3.2.2.2 Benefits

- Fast delivery of the software.
- Customer satisfaction.
- Allows adoption of enhancements even at later stages.

3.2.2.3 Why the project can not employ it?

All Agile methodologies are team oriented. Apart from not having a team, an iterative approach is still required by the project because of it's ever changing requirements in the products it uses as also mentioned in section 3.2.1.4. So iterative approach of agile methodologies is suitable for the project. But it's team work focused practices can not be implemented due to resource constraints (both time and people working on it).

3.2.3 Rational Unified Process

Rational Unified Process shortly called as RUP (and references as RUP here onwards in this document) is both a process and a product. RUP is a process product. It is developed by Rational Software and it comes with a suite of development tools. Kruchten (Kruchten 2003) says that RUP is also a process framework which is adapted and often extended to suit the organisational needs. Fowler (Fowler, M 2000)says that RUP provides a common set of practices and development teams can choose those practices that suit their projects. RUP is use case driven, iterative and architecture centric. Fowler (Fowler, M 2000)also says that RUP is adapted in infinite number of variations ranging from Waterfall woth 'analysis iterations' to picture perfect agile.

3.2.3.1 RUP Architecture:

The two dimensional architecture of RUP is shown in the figure3.2. Time is represented by the horizontal axis which shows the lifecycle of the process and workflows of the core processes are represented by vertical axis.

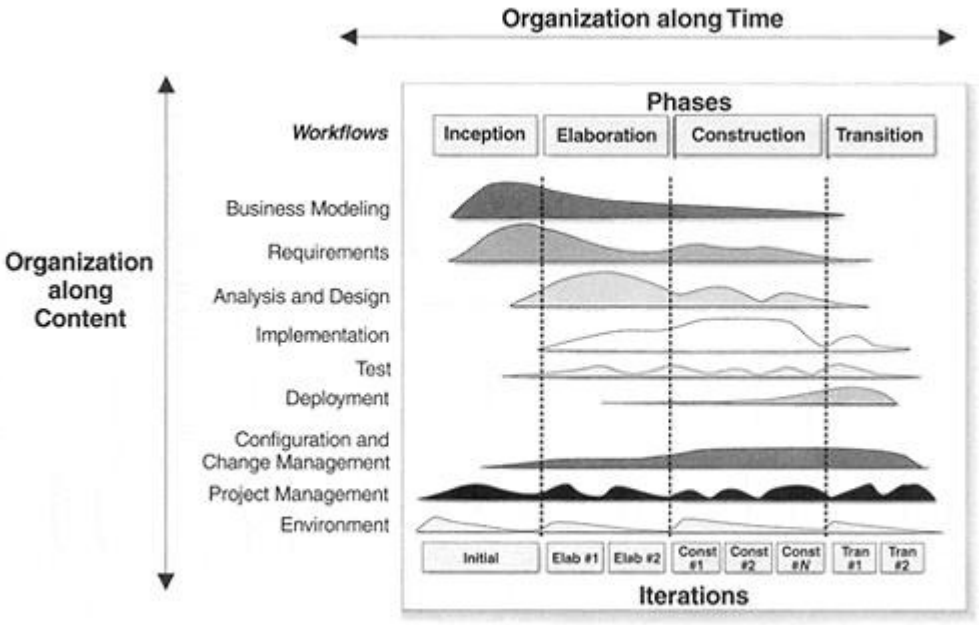


Figure 3.2: Process Structure of Rational Unified Process (source: Kruchten (Kruchten 2003))

The phases of RUP are described as below:

- **Inception:** The project is evaluated typically by stakeholders to decide if it is viable to do the next phase.
- **Elaboration:** Use cases are identified in this phase and the software is developed in iterations. At the end of this phase, a skeletal working system should be developed that would act as a starting point for development.
- **Construction:** Functionality of the software is developed in this phase.
- **Transition:** Activities done at later stages are included, like deployment and training, but may not be done iteratively.

3.2.3.2 Reasons why RUP suits the project:

- The support for SN applications might be changed very often and so do the requirements. So an Iterative method is more suitable.
- In RUP, integration is not one big task done at the end, instead, it is done progressively. So the approach is a process of continuous integration.
- The iterative approach in RUP lets the developers mitigate risks earlier as generally more risks are discovered and addressed at integration time. So risks discovered in each iteration are addressed in the same iteration.
- Common parts are designed and then implemented partially in the iterative approach and this eliminates the need for identifying all the commonality in the starting phase itself before anything is designed.
- RUP mainly consists of models, developed and maintained constantly, of the system being developed. According to Kruchten(Kruchten 2003), the models help the architects and developers to understand the problem and shape it's solution. Models are created or written using UML (Unified Modeling Language). But UML does not tell the developers how to develop software. It just provides the required vocabulary but does not essentially help in writing the book. RUP comes along with the UML to complement it's processes and modeling them. Hence it is a reasonably good approach to use RUP for developing small or large softwares.
- The approach of RUP is use case driven but not process driven and hence use cases defined for the system become the basis for the development process at any stage.

3.2.4 Methodology Adopted

The RUP methodology is not employed as is(with it's two dimensional approach) but is adapted as required. The Business Modeling, Requirements, Analysis are done in non iterative manner. Design and Implementation phases are done in iterative manner where each iteration adds a use case and then tested. Essentially the adapted methodology resembles Waterfall model except for Design and Implementation phases along with some testing done in iterations. For each SN integration, the whole methodology is applied again which means that it is started from analysis and optional design followed by iterative implementation, testing and evaluation phases.

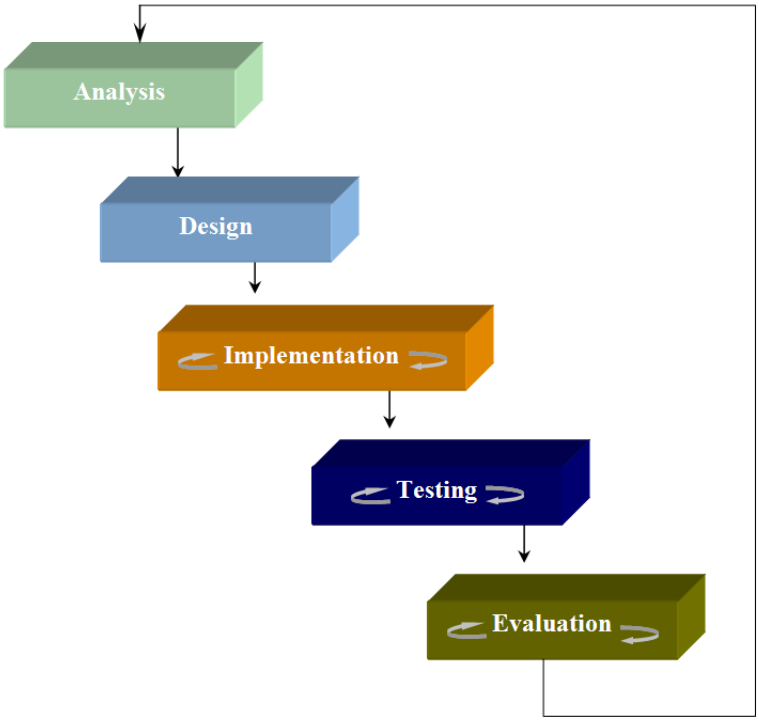


Figure 3.3: Development methodology used

3.3 Conclusion

After analysing the process, benefits and shortcomings of the waterfall, agile and RUP methodologies, this chapter discussed why they can not be adopted in the project as they are. As many practices of RUP suit the project, they are adopted but not as they are.

The adopted methodology for this project is a hybrid form of waterfall and RUP methodologies as explained in the section 3.2.4. Havind decided on the methodology to be used, the development life cycle is covered in the rest of the documents.

Chapter 4

Analysis

4.1 Introduction

The development methodology is explained in the chapter 3 and the project is analysed in this chapter as a first phase of it's life cycle. The requirements are first analysed and then a business model is presented along with the system analysis model. These models form the basis for the system design. The models are the UML diagrams created using Rational Rose and hence the chapter is organised based on Rational Rose views.

4.2 Requirements Analysis

The requirements are finalised with the help of a survey conducted for a group of users.

4.2.1 Survey Design

For the requirements analysis, a survey is conducted with simple set of questions as to what is expected in general for a product that integrates IM and SN systems and does it really interest the people at work and at home. The survey was just user oriented and not technology oriented. About 25 users at work and 25 users at home were chosen for the survey. They were asked a set of questions that help in revealing the requirements of the project.

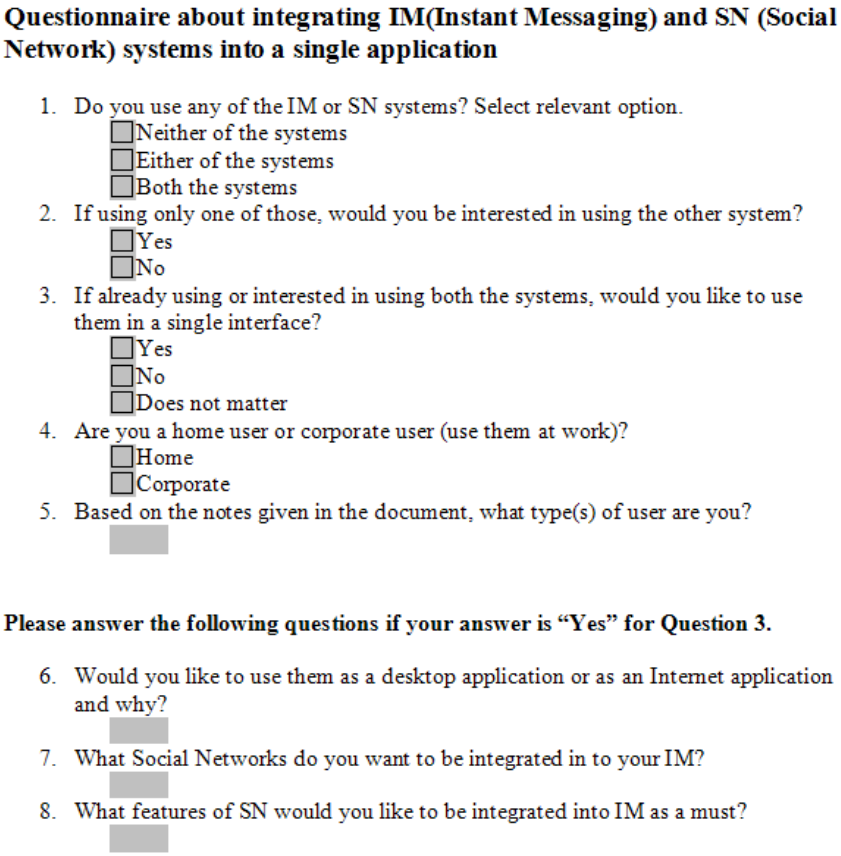
4.2.2 Survey Questions

List of questions asked is shown in the figure 4.1. There were two kinds of questions. One is multiple choice questions and the other is descriptive.

Questions 1 through 5 were designed to get a rough estimate of type and amount of people interested in using this software and thus to know if this product would be useful at all. Questions 6 through 8 were designed to know the preferences of users for the integration.

Question1,2 These questions were designed to obtain a rough estimate of people, who would have interest in this software. People who were already using or interested in using both IM and SN systems, would be the parties interested to use this product.

Question3 This question was designed to know if the users would like to access both systems in single interface.



Questionnaire about integrating IM(Instant Messaging) and SN (Social Network) systems into a single application

1. Do you use any of the IM or SN systems? Select relevant option.
   - ☐ Neither of the systems
   - ☐ Either of the systems
   - ☐ Both the systems
2. If using only one of those, would you be interested in using the other system?
   - ☐ Yes
   - ☐ No
3. If already using or interested in using both the systems, would you like to use them in a single interface?
   - ☐ Yes
   - ☐ No
   - ☐ Does not matter
4. Are you a home user or corporate user (use them at work)?
   - ☐ Home
   - ☐ Corporate
5. Based on the notes given in the document, what type(s) of user are you?

Please answer the following questions if your answer is "Yes" for Question 3.

6. Would you like to use them as a desktop application or as an Internet application and why?
7. What Social Networks do you want to be integrated in to your IM?
8. What features of SN would you like to be integrated into IM as a must?

Figure 4.1: Questionnaire

Question4 This question was designed to help analyse the results of the questions 5,6 and 7, as home users might have different preferences of how they would like the resulting integrated software to be and corporate users might have different preferences.

Question5 This question was designed to know what type of users and thus know their attitude towards Social Networks. These results help to analyse the usefulness of the software in evaluation phase covered in chapter 8.

Question6 This question was designed to know the type of interface users prefer, either internet application which can be accessed through browsers or desktop application which does not need a browser.

Question7 This question was designed as an other way to know the popular sites used among the sample users.

Question8 This question was designed to get the list of features that most people generally use. There can be many features supported by different SNs but this question helps starting the project with most wanted features and add any other features in future versions.

4.2.3 Survey Outcomes

Out of 50 sample users, 35 were both IM and SN users who also wanted to have a single desktop interface for IM and SN systems. The reason as analysed is that some of the users at work do not have access to many of the social networking sites. Many of the corporate IM users, who

have access to SN sites, and most of the home users do not like to keep all the SN sites open and constantly monitor for their friends, to see if they are online and that the desktop application is easy and flexible to use than internet application. The interesting part is about which SN sites they use. The results were charted as a bar diagram shown in figure 4.3. The top two sites were Facebook and Bebo. 84% of the people use Facebook (Facebook and others), 72% among them use Bebo (Bebo and others).

The percentage of SN users of each type, as categorised in section 2.2.3.2 , out of the sample users are charted in picture 4.2. More percentage of users (both home and office) are Faithfuls and Followers. Though this project does not interest all Followers (The people who joined the social sites just because others were doing or to keep up with the changing technologies or to keep up with their peers), it would interest some of the Followers. It would interest people who want to use SNs for the sake of keeping up with others but not for keeping up with changing technologies, as this integration might or might not support all the services that SNs support.

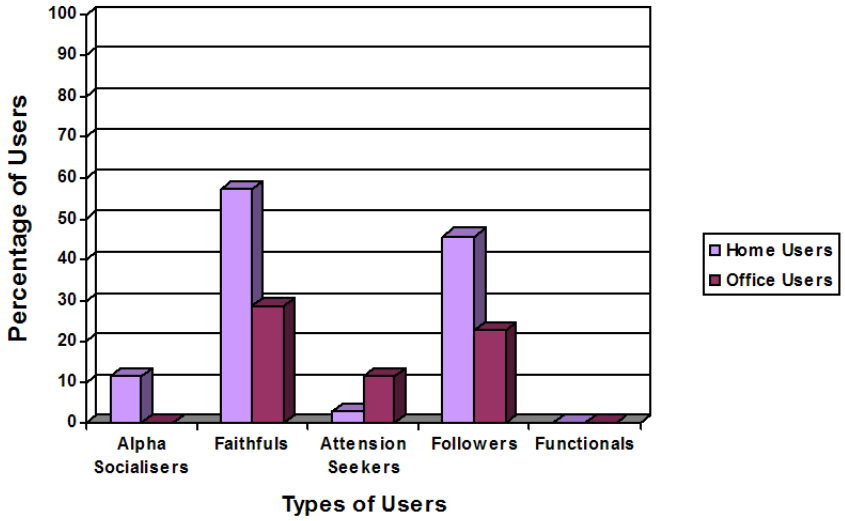


Figure 4.2: Bar diagram to show the percentage of types of users

Majority of the users at work did not want all features of the SNs for integration but just some of the common features like

- Online presence
- Maintaining friends list
- Chatting
- Sending and receiving files for collaboration
- Sending and receiving emails

Some of the home users wanted all possible features for integration.

The requirements gathered are represented as use cases and modelled in section 4.3.

4.3 Analysis Modelling

UML diagrams for the project are drawn using Rational Rose and so the diagrams are organised by the views as laid out by Rational Rose. In Rational Rose, the model is constructed based on

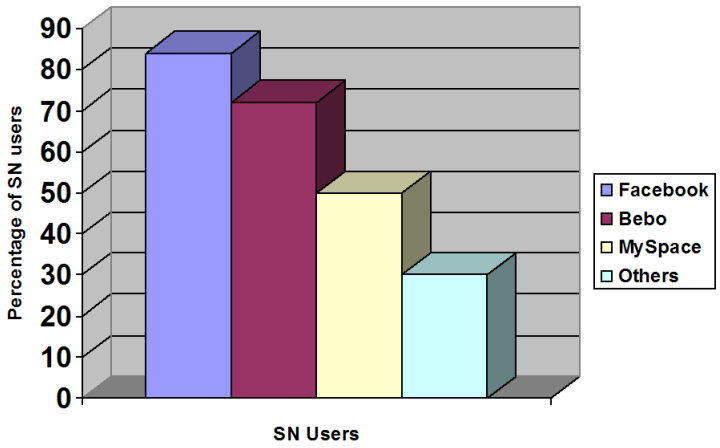


Figure 4.3: Bar diagram to show percentage of social network users

the type of project and the template used for this chapter is of Java.

4.3.1 Use Case View

In Rational Rose, business use case model and system use case model are organised in use Case view as shown in figure 4.4.

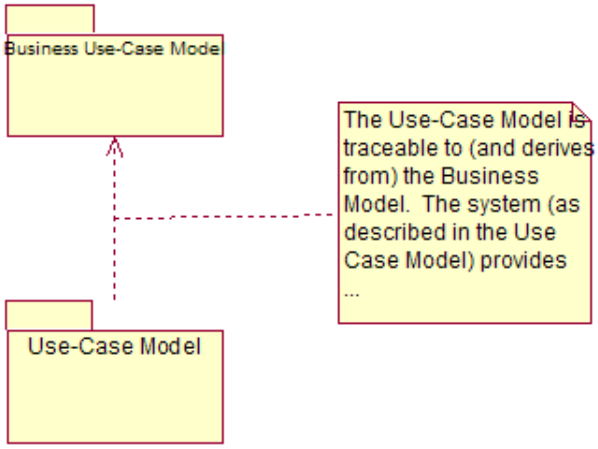


Figure 4.4: Use case view

4.3.1.1 Business Use case model:

In business use-case model, each business use case represents a business process from the view of an external user without any insight into the application. It just represents what can be achieved by the project. The business use case represented in the figure 4.5 depicts the business user who is a user of both Instant Messaging system and Social Networking sites. The diagram only presents some of the usecases for following reasons:

1. The IM related use cases provided by the project will depend on the IM system chosen at later stages. However IBM Sametime is proposed in the Chapter 1 and it provides many usecases of which three are shown in the diagram.

2. The SN use cases provided by the project will depend on the social network which is being integrated with IM. However common usecases like GetFriends, Chat with Friends are presented in the diagram.

4.3.1.2 Use Case Model

The use case model in RUP presents a model which supports business processes and thus describe the behaviour of the system. Figure 4.6 is the simplified use case model of the project. It is simplified for the reasons given in the section 4.3.1.1. The actor in the use case model is termed as "SNIMUser" and the name is self explanatory and means that the user named by it is a Social Network and Instant Messaging user. This SNIMUser actor realises the "User" actor in Business use case model.

4.3.2 Logical View - Analysis Model

This View presents the analysis models. Analysis model consists of the analysis classes that describe an abstract realisation of all the system use cases using different kinds of diagrams including sequential diagrams, class diagrams and activity diagrams. Key classes are identified and the brief function of them is identified. Boundary classes, control classes and entities are modelled and related in class diagrams. The analysis model then evolves into design model. Figure 4.7 presents the analysis model structure of the project.

4.3.2.1 Preferences Module

For user friendly integration, the project provides some preferences that users can set as per their requirement. For example, the project proposes to support the general preferences like if the integration is needed for a social network, the IM group(group name) to which friends from that particular social network can be added. Figure 4.8 presents the class diagram for preferences module. The main classes as designed at this stage are

PreferenceVault: This class represents the place holder for preferences. All the preferences either default (when user do not select any) or user set ones are stored here for later use.

PreferenceManager: This class manages the storage and retrieval of the preferences and hence depicted as control class.

Preferences: This class represents the preference that is being set and stored or retrieved and hence depicted as entity class.

PreferencePage: This class can be a delegate or the UI (User Interface) itself that is presented to the user for setting the preferences and hence depicted as boundary class.

IPreferenceManager: This interface is exposed for the SN implementation module so that it can implement it's own logic of interpreting the preferences and/or provide customised preferences that are relevant to the Social Network code that it manages.

4.3.2.2 Integration Module

This module provides the basic framework and interfaces required for the Implementation module. Integration module is the generic and essential module to all social network implementations where as Implementation module is specific to a single Social Network. The structure of this module is shown in the figure 4.9.

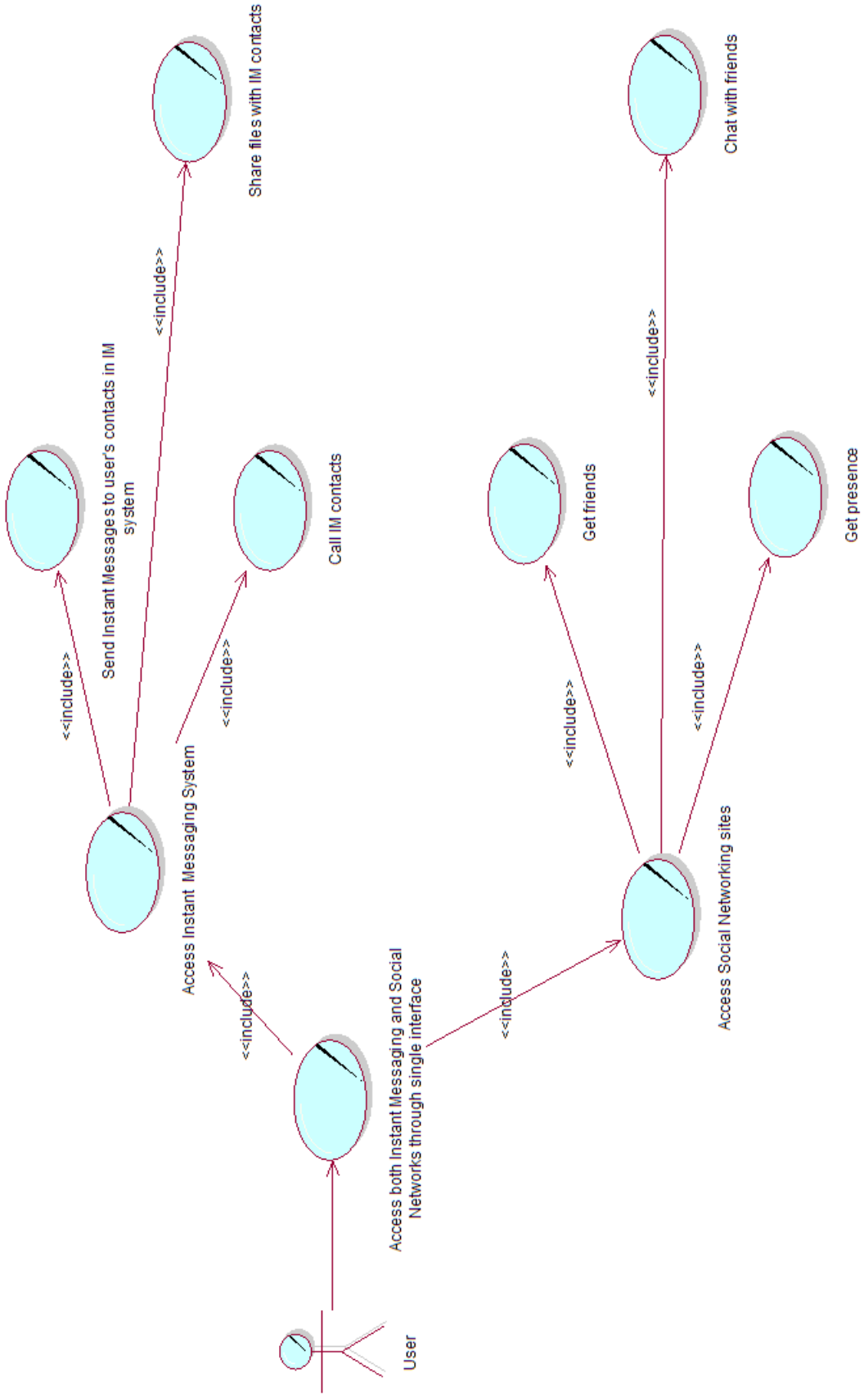


Figure 4.5: Business use case model

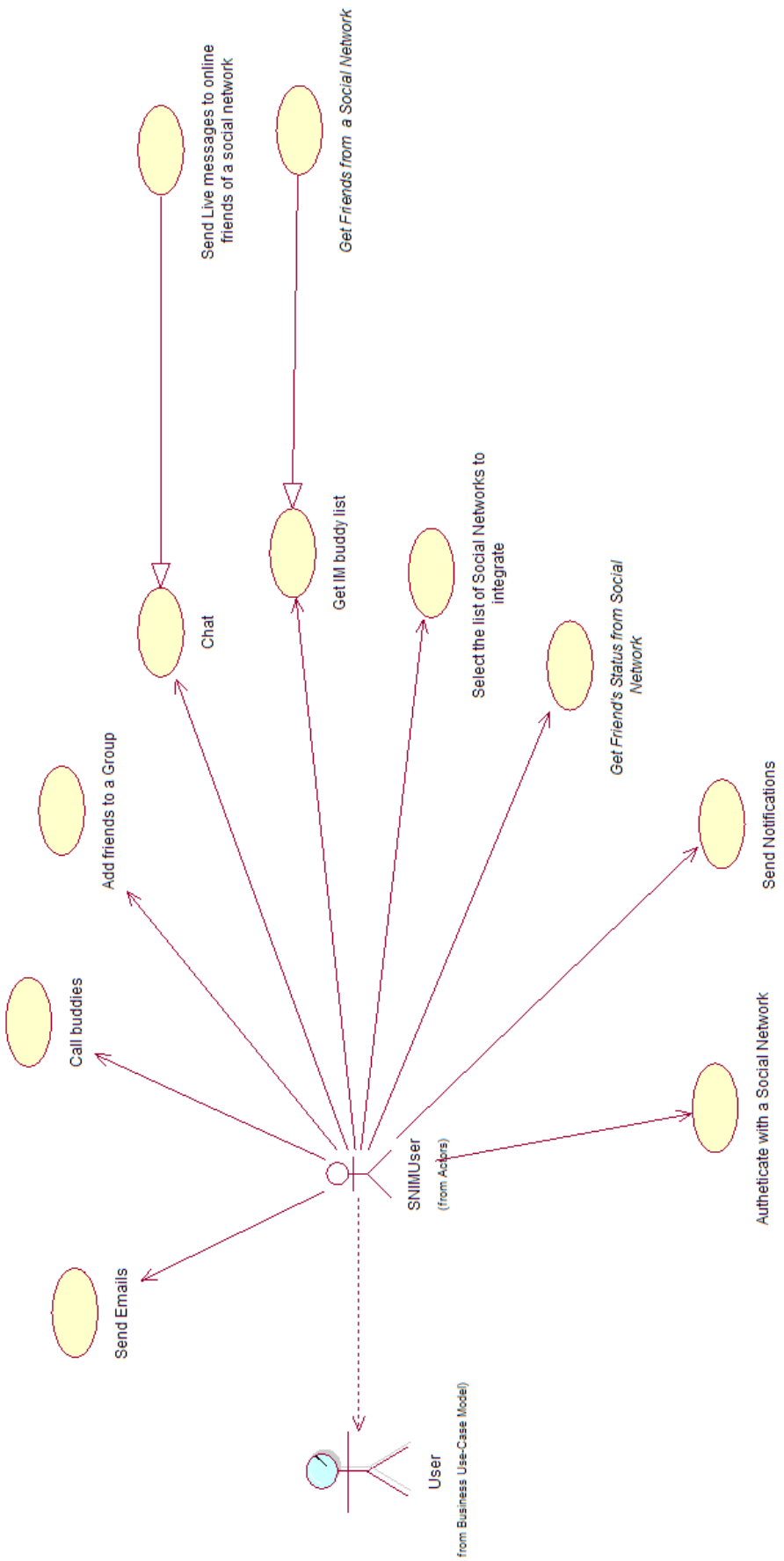


Figure 4.6: Use case model

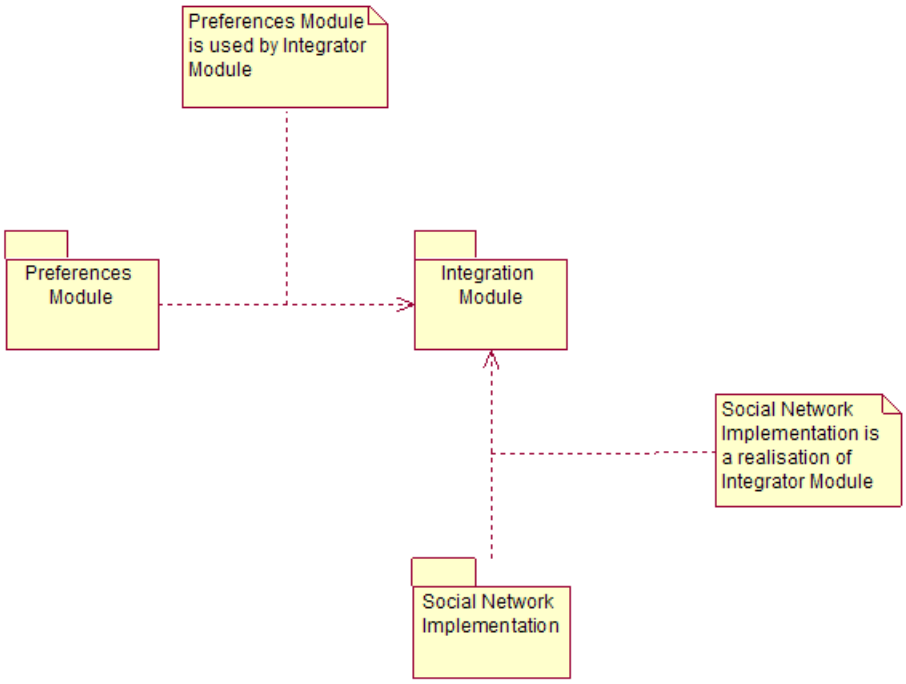


Figure 4.7: Analysis model structure

1. **General Actions for supported Social Networks:** This sub module contains the UI required for the integration of IM and SN systems. This UI for Sametime can be a menu in the toolbar. This generic actions can lead to different UI for each of the social network that is integrated. Figure 4.10 depicts the class diagram for this sub module. IBM Sametime provides some extension points which allow applications to hook into Sametime UI as part of Eclipse framework. The plugins and extension points are explained in detail in section 6.3. In the class diagram depicted in figure 4.10, "IM UI extension point" represents the extension point provided by Sametime API. The application proposed by the project(referred as the current application or this application here onwards in the document) extends or defines this extension point to provide the UI for all SNs selected for the integration as a list of menus for example where each list item can be a menu for actions related to each of the SNs. "Generic Menu for Social Networks supported" represents the class that extends the extension point " IM UI Extension Point". This class provides a means to display the UI for actions organised by the list of Social Networks. In simple implementations this "means" can be a list of menus each for one SN and that menu provides menu items for that particular SN and thus, "UI for single SN" will be list item or sub menu provided by the "Generic Menu". This is an interface because this should be implemented by the SNs that need to be integrated. Similarly, the interfaces like "Login" and "Logout" depicted in the figure 4.10 will be the menu items which represent actions for that particular SN. The list of actions provided at this level depends on the availability of SNs API. Basically the design is to provide menus one for each SN that needs integration and that menu provides the actions relevant to that SN. This is the design followed across all the UI extension points used in the project.

2. **Group and Person Actions:** This sub module contains the UI for the actions as ex-

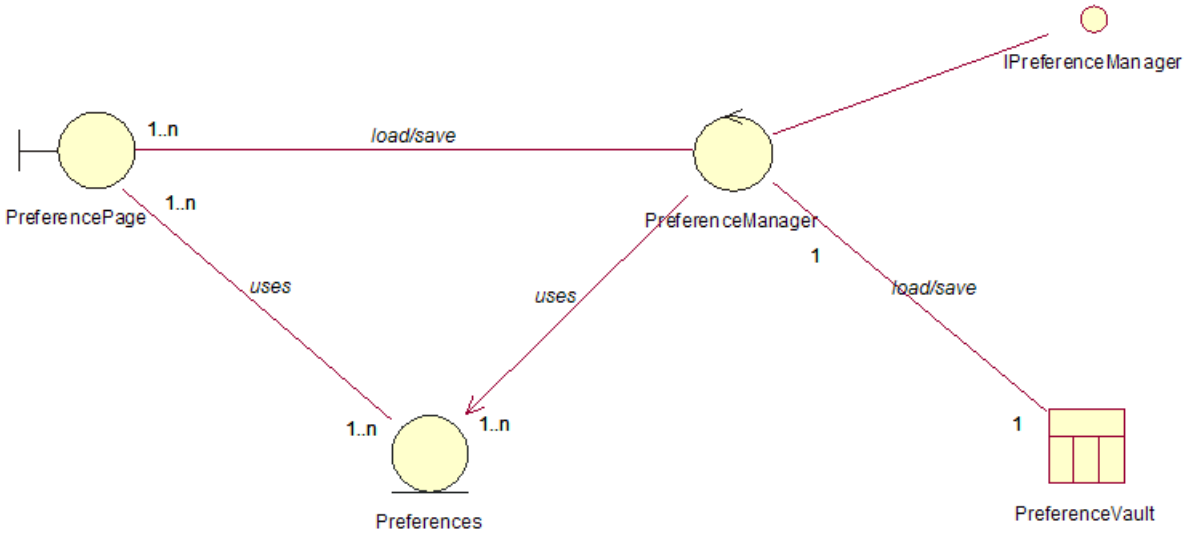


Figure 4.8: Class diagram for Preferences module

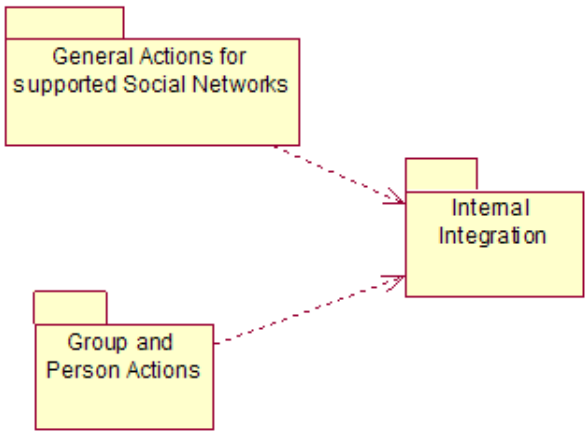


Figure 4.9: Integration Module Structure

tensions to group or person objects of the IM. For Sametime, these can be context menu actions for group and person objects(in buddylist) which can be seen on right clicking on a group or person. Figure 4.11 is the class diagram for Group and Person Actions sub module. In the figure 4.11, the "Group Action Extension Point" class is an extension point provided by Sametime to add actions to groups. The applications that extend this extension point can add custom actions to the context menu of a group in the buddy list. The "Generic Menu for an SN" is a menu inside a context menu of the groups, which provides the relevant actions for that particular SN. Same is the design for Person actions. Person actions are added to the context menu of a person in the buddy list.

3. **Internal Integration:** This module contains the relevant code to perform the main integration. It is a set of classes and interfaces, which when used by the Implementation module, manages the user actions, acts as middle layer between the UI and SN client. It receives the actions requests from UI which it then processes along with the data from preferences, SN Registry and other entities to generate SN client requests. It again receives

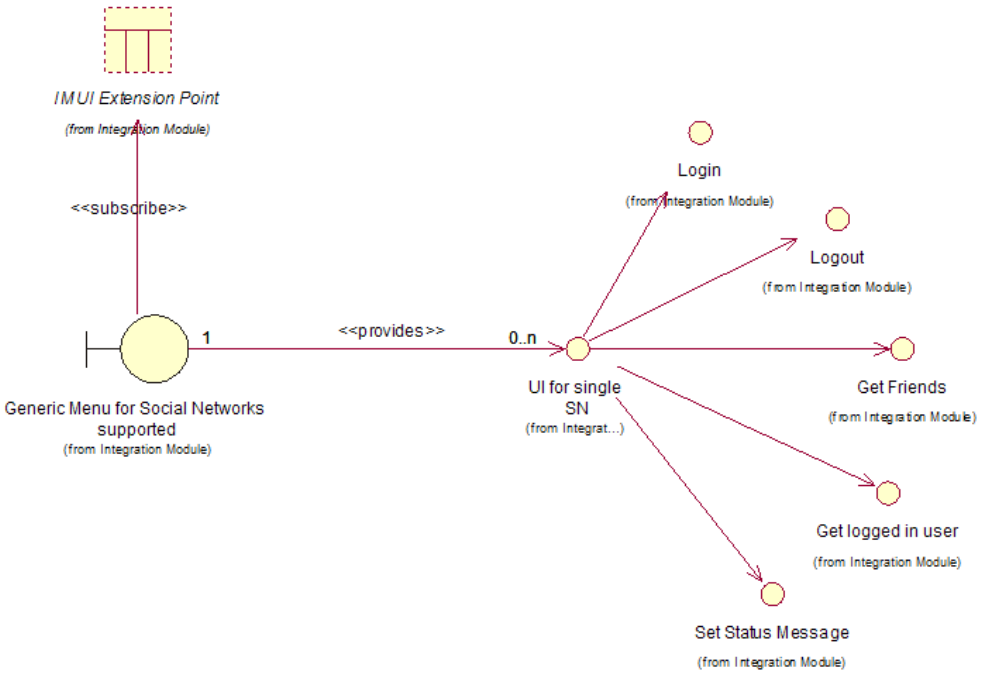


Figure 4.10: Class Diagram for General Actions

the responses from SN client, matches these responses with the IM formats and then presents them to the user. The figure 4.12 depicts the relations between the classes and interfaces used in this module.

Activator: It is part of the plugin architecture discussed in sub-section 6.3. This class activates the whole plugin. It's only after the activation that the plugin responds to all kinds of events.

SNCommunityListener: It is part of the Sametime which listens to the community events and thus extending this provides an opportunity to react on the occurrence of community events.

ManagerFactory: It is part of the Factory design pattern which is defined in sub-section 5.4.2.

SNManager: ManagerFactory creates and destroys the objects of this type. This can be extended by the specific SN module. This handles all the activities needed for the integration.

FriendsList: It stores the list of friends (contacts from an SN).

SNSettings: It contains all the settings required for communications with the SN. It's simplest implementation can be a file with all the properties set.

SNUser: It is a generic class that represents a user of SN. This class when extended in the SN implementation module, represents the user of that particular SN.

SNUserManager: It manages the list of SNUser type objects. The simplest implementation is to map person objects(Sametime's user) to corresponding SNUser objects.

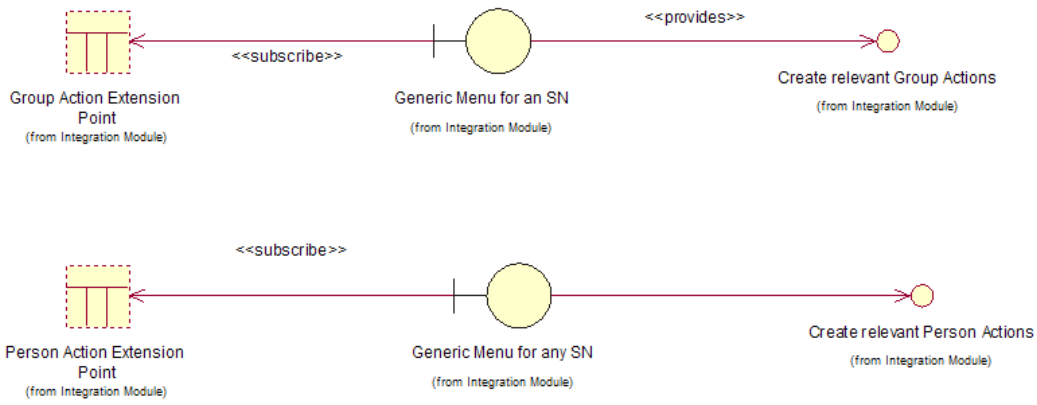


Figure 4.11: Class Diagram for Group and Person actions

4.3.2.3 Social Network Implementation Module(SN implementation module):

This module, as depicted in figure 4.13, realises the modules from Integration module. This module contains the code for one particular SN. So for each SN that needs integration, the relevant code must be in this structure.

1. **Realisation of General Actions:** This sub module, as the name says, provides the realisation of the interfaces laid out by "General Actions" sub module 1 of Integration module. The figure 4.14 is the class diagram for this module.

   Specific Menu for a Social Network: This is the specific menu for one SN which provides menu items for generic actions relevant to the corresponding SN.

   Authentication Module: This class performs the authentication tasks for that specific SN. This also uses the "LoginForm" to present an UI for authentication credentials.

   Specific SN Proxy Client: This class is a proxy for the SN specific client and communicates with the corresponding "SN API".

2. **Realisation of Group and Person Actions:** This sub module, as the name says, realises the interfaces laid out by "Group and Person Actions" sub module 2 of Integration module. The figure 4.15 is the class diagram for this module. It represents the involved classes and the relation between them.

   SN relevant Group Menu Creator: This is the UI that is presented to the user as Group action menu for the corresponding SN. This class implements the "Create relevant Group Actions" interface that is provided by the "Group and Person Actions" (sub section 2) module in Integration module. The relevant actions are passed down to "Specific SN Manager" from "Social Network Proxy" module.

   SN relevant Person Menu Creator: This is the UI presented to the user as Person action menu (the context menu for the person in buddy list). This class implements the "Create relevant Person Actions" interface provided by the "Group and Person Actions" (sub section 2) module in Integration module. The relevant actions are passed down to "Specific SN Manager" from "Social Network Proxy" module.

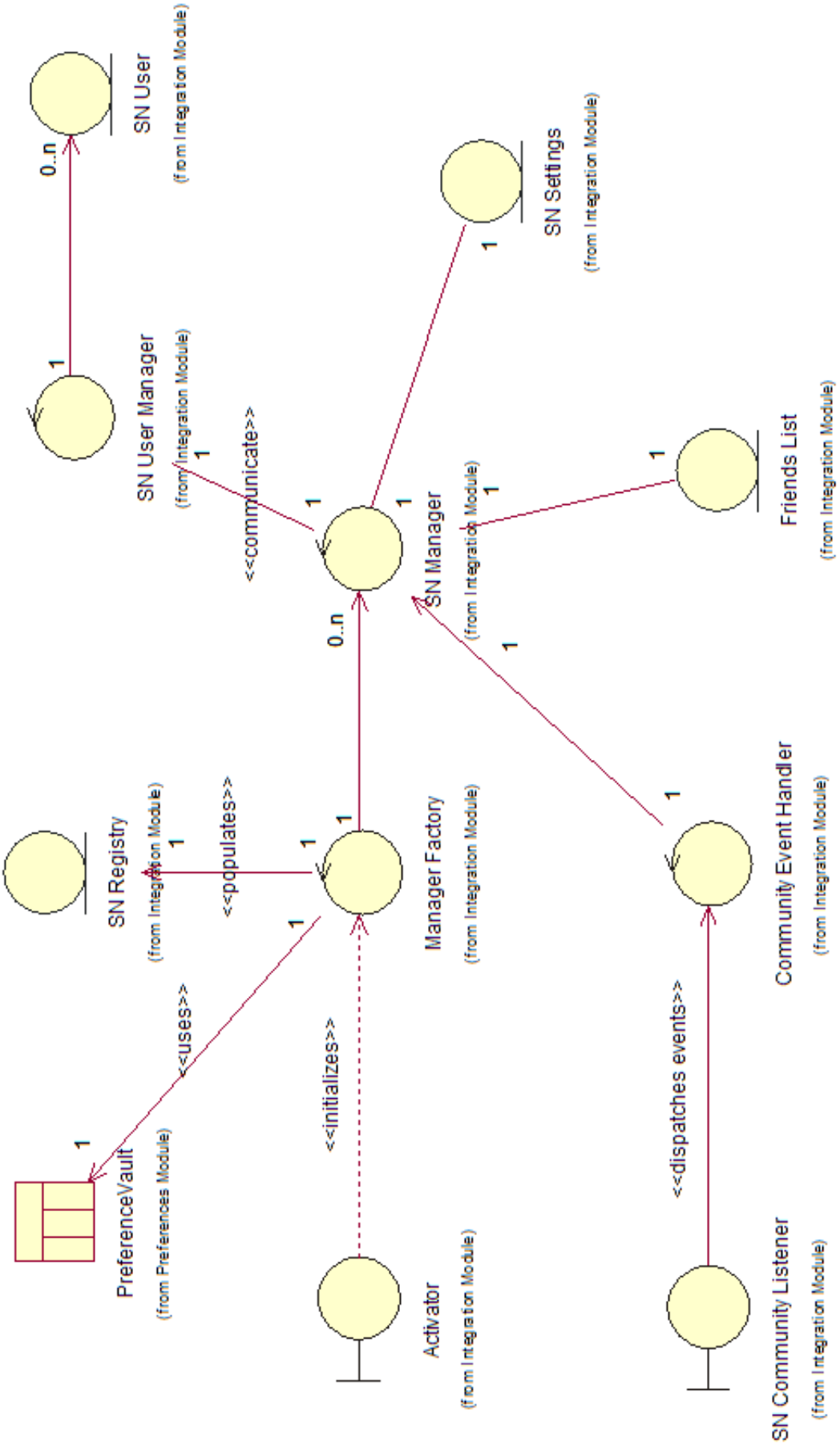


Figure 4.12: Class diagram for Internal Integration module

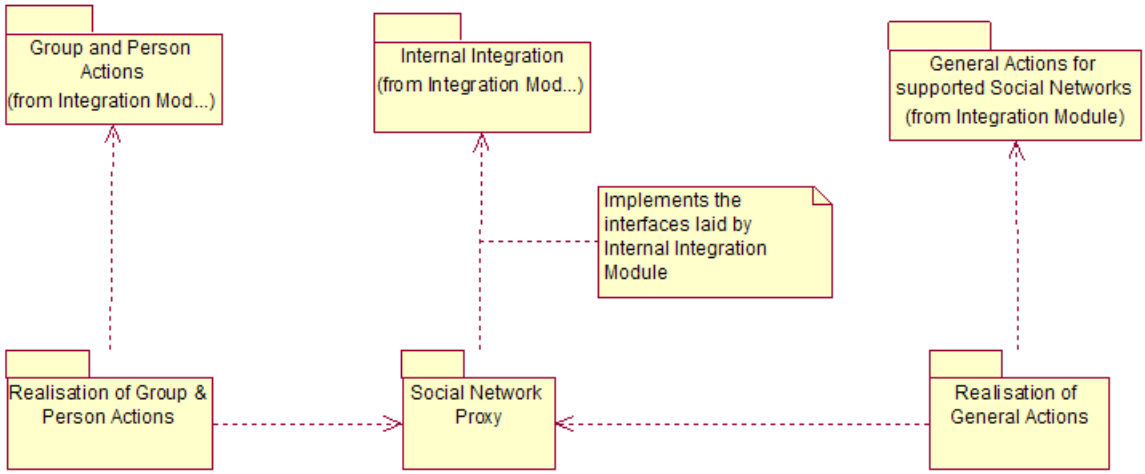


Figure 4.13: Social Network Implementation module structure

3. **Social Network Proxy:** This sub module contains the classes that support the code that is required to act as a proxy for the "SN client" from "SN API". The actions from the UI are passed down to this module. This module is responsible for the actions to be processed and interpreted and sent as requests to the SN. The figure 4.16 is the class diagram for this module.

   Specific SN User: This is the class that represents a SN specific user and this extends the "SN User" class from Integration Module.

   Specific User Manager: This class manages the SN specific users and maps with their unique ids and Person objects(Sametime persons) for the use of all modules mainly SN Manager. This class extends "SN User Manager" from Integration module.

   Specific SN Manager: This class extends or generalises "SN Manager" from Integration module and performs the correlation tasks as described for "SN Manager" but for a specific SN.

   Specific SN Proxy Client: This class acts as a proxy to the actual SN specific client available from "SN API".

   SN API: This package is not developed as part of this project but is used as part of the library. This package provides the actual SN specific client and it's supporting classes.

4.3.2.4 Use case traceabilities:

This section presents the use case realisations for the use cases laid out in section 4.3.1.2 and the sub diagrams to explain some of the important use cases. The figure 4.17 shows the use cases that realise the use cases from use case model.

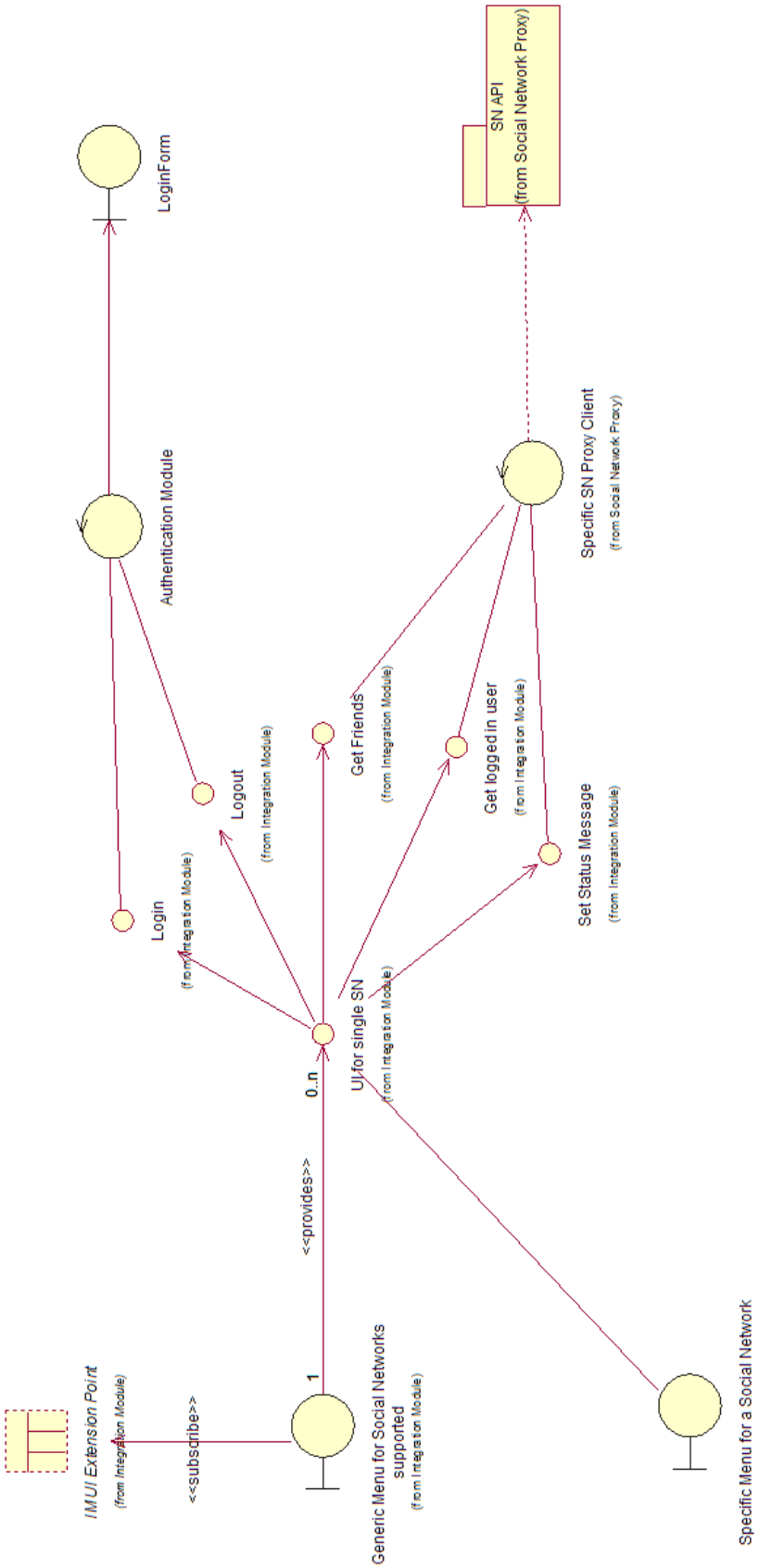


Figure 4.14: Class diagram for Realisation of General Actions Module

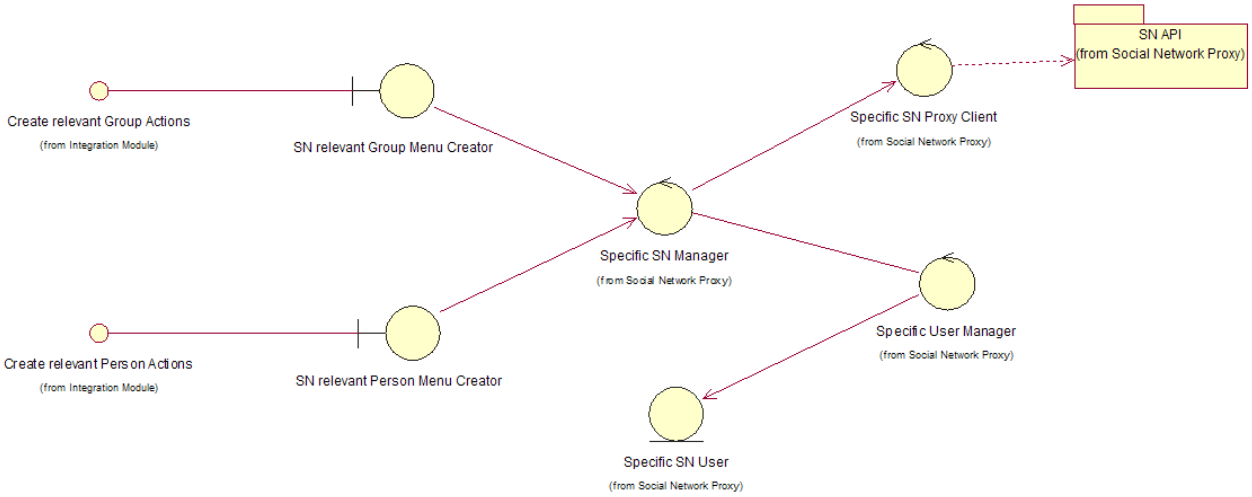


Figure 4.15: Class diagram for Realisation of Group and Person Actions module

Table 4.1: Type of use cases

IM feature	SN feature	Application
Send Mails	Send Live Messages to Friends	Select Social Networks to integrate
Call Buddies	Get Friends from a Social Network	Get Online Presence of Friends
Add friends to a group	Send Notifications	Authorise the Social Network Application
Chat		

The use cases can be grouped into three based on which product offers the features they describe, if it is IM feature or SN feature or that is offered by the application (project) or related to the integration of SN and IM. Among the use cases presented in the figure 4.17 , the table 4.1 shows the classified use cases. This section presents the detailed diagrams for atleast one of the use cases in each type (types shown in the table 4.1). From here onwards, the application or software developed as part of the project is referred as "Application" and the social Network application is referred as "SN application" or just "social network application" itself.

1. **Use case - Authorise the Social Network application**

   Brief Description: This use case allows the user to authorise the social network application (Any Facebook application must be authorised by its owner before it can be connected to the Facebook. This plugin also uses a Facebook application created specifically for the purpose of the project and the application page at Facebook is shown in Appendix D).

   Events: The events involved in this use case are clearly explained using a sequence diagram as in figure 4.18

   (a) User clicks the corresponding menu item to login.

   (b) The application opens the login form.

   (c) User enters the login credentials.

   (d) The application sends the credentials to the social network and gets the session information.

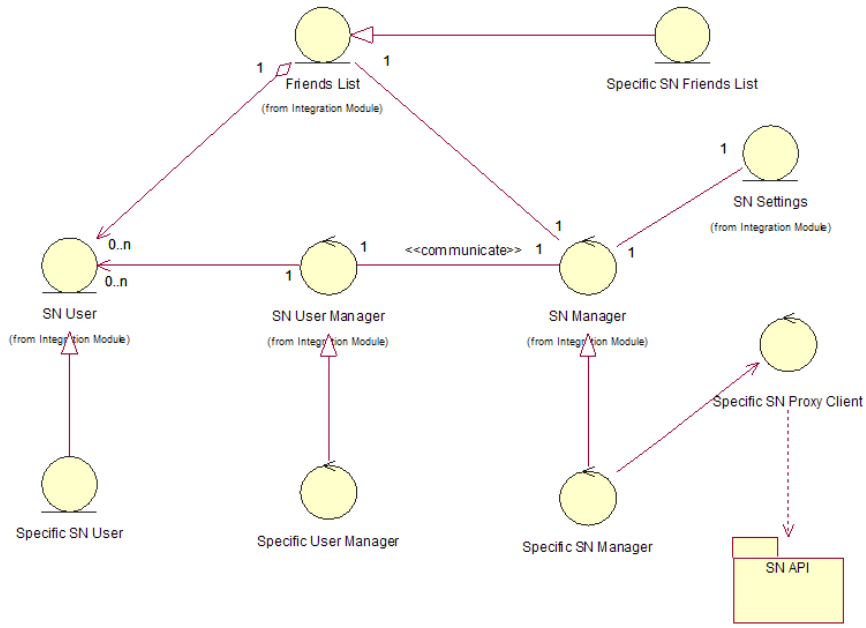


Figure 4.16: Class diagram for Social Network Proxy module

Preconditions:

(a) The social network application must be configured properly. It should be created at the SN website and thus should have the application details like api-key and api-secret, which are required for the communications with SN and must be stored in the configuration file.

(b) The user currently trying to login must be the user of the SN.

Postconditions: After the application is authorised by an authentic SN user, any of the startup tasks can be executed.

The figure 4.19 shows the activity diagram for this use case along with post login activities in general. The application owner or creator creates the application at the SN website. The Implementation module must know the settings or details of the application like api_key and api_secret as already mentioned in Preconditions of this usecase and should store those values in the configuration file. The post login actions or startup tasks referred in postconditions of the use case are shown in this activity diagram and they differ from SN to SN but in general, they can be the tasks like Loading friends form SN and add them to the IM buddy list, Get the status of all friends in a separate thread if getting friends does not include this information and track the session time out . This application being only client side desktop application, it may or may not be able to listen to the events sent by the server. In such cases, the session time out must have to be simulated.

2. **Use Case - Send Notifications**

Brief Description: This is an example of a feature offered by a social network. This allows users to send notifications to other user(s) of the SN application

Events:

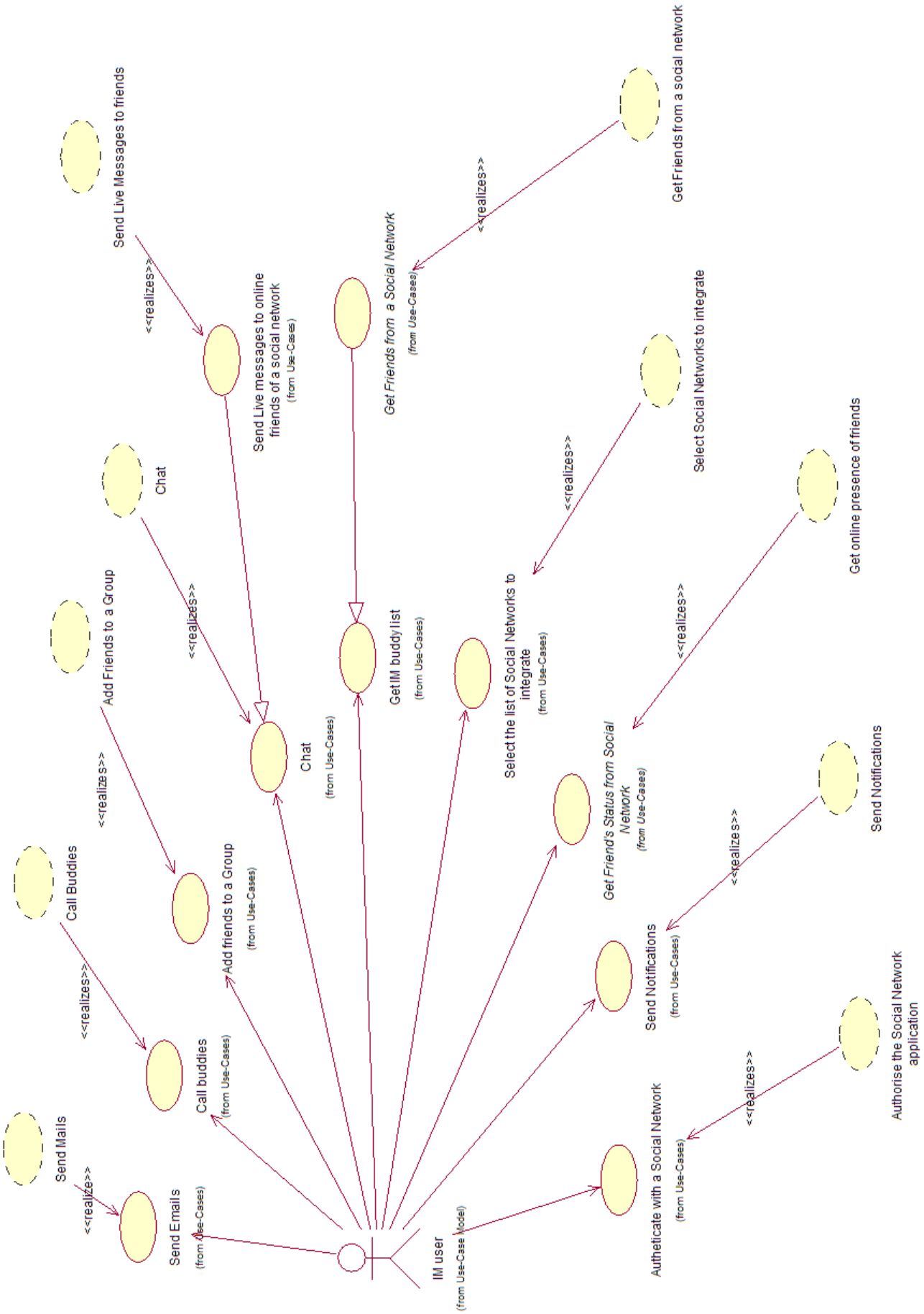


Figure 4.17: Use case Traceabilities

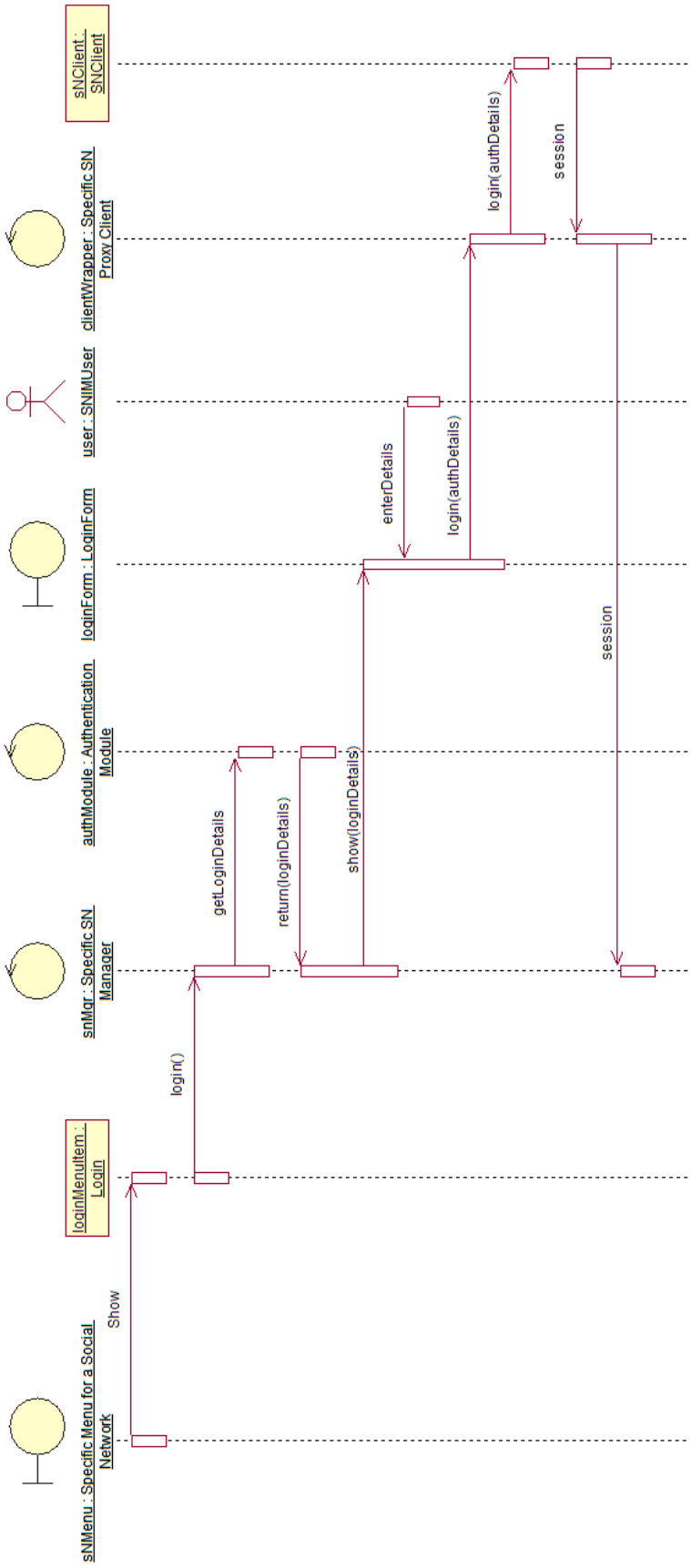


Figure 4.18: Sequence diagram for "Authorise the Social Network application" use case

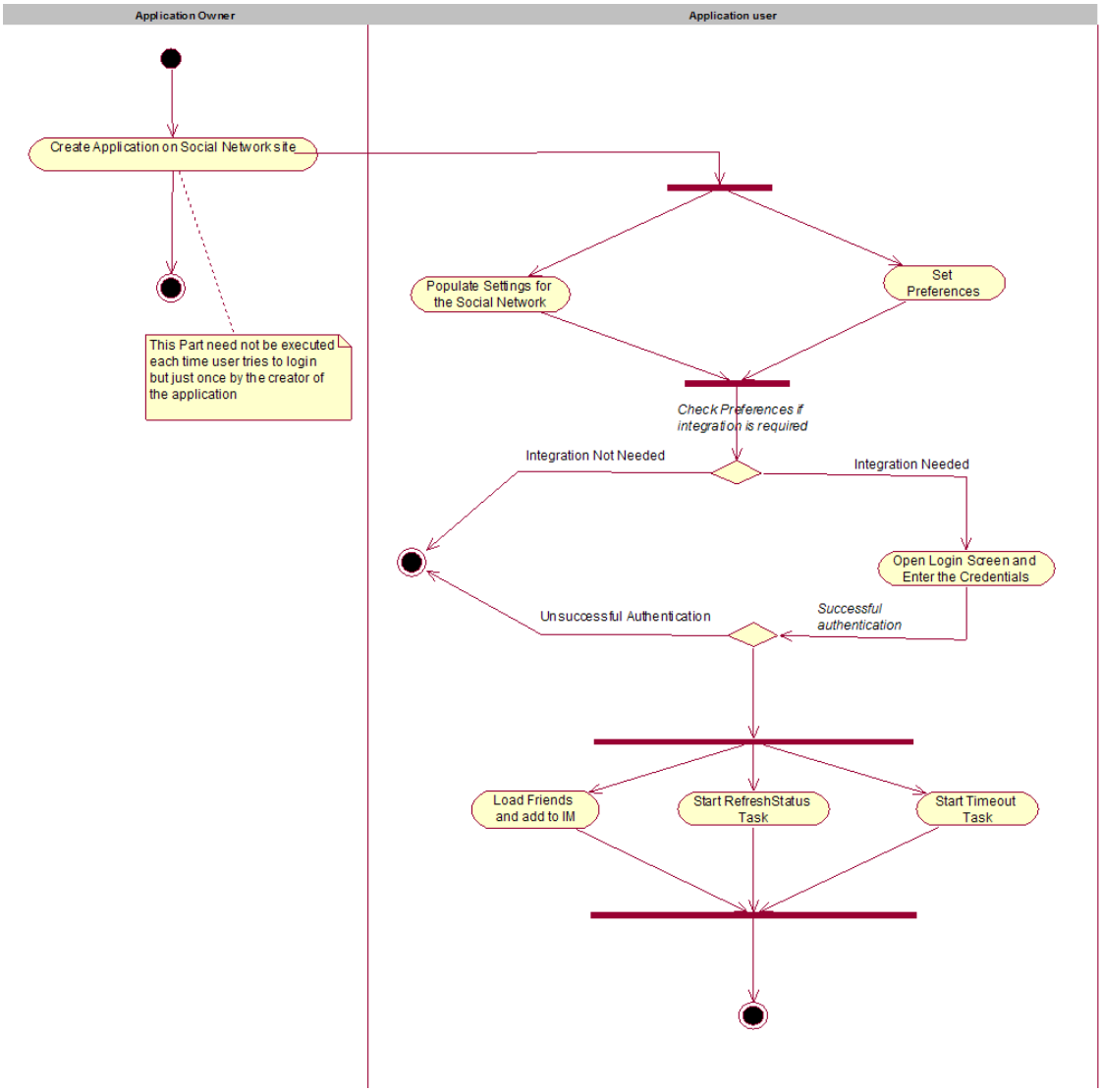


Figure 4.19: Activity diagram for login use case and post login activities

(a) Users trigger the action relevant to Send Notifications by selecting the corresponding UI.

(b) Application processes the action and presents the users with a form or window which asks for the required data before sending the notifications.

(c) Users enter the required data.

(d) Application wraps the data into format that SN client requires and sends the request to the client.

(e) SN client sends the request to the server.

(f) Server sends the response to the SN client.

(g) SN client receives it and forwards to the application.

(h) Application interprets the response and if there is any error, shows the relevant reason of failure to the users.

(i) In case of users not having relevant permissions for the "Send Notifications" action, and if the SN allows a way to ask them if they want to grant permissions, then the application shows Grant Permissions dialog to the users.

(j) Users response is then processed.

(k) After the user has granted permission in the last step, they can initiate the action again.

The sequence diagram in figure 4.20 depicts the events described above.

Preconditions: The user must have authorised the application by logging in.

Postconditions: If no errors occurred at server, the notifications will be sent, if there is any error, the user will be notified.

3. **Use case - Call buddies**

Brief Description: This feature is offered by the Sametime IM used for the project. It allows the users to call the contacts in the IM's buddy list.

Events:

(a) User starts the action by selecting the UI relevant to calling buddies. One of the user interfaces is the context menu for buddies (person objects) in case of ST.

(b) The call is initiated and user can talk to the callee.

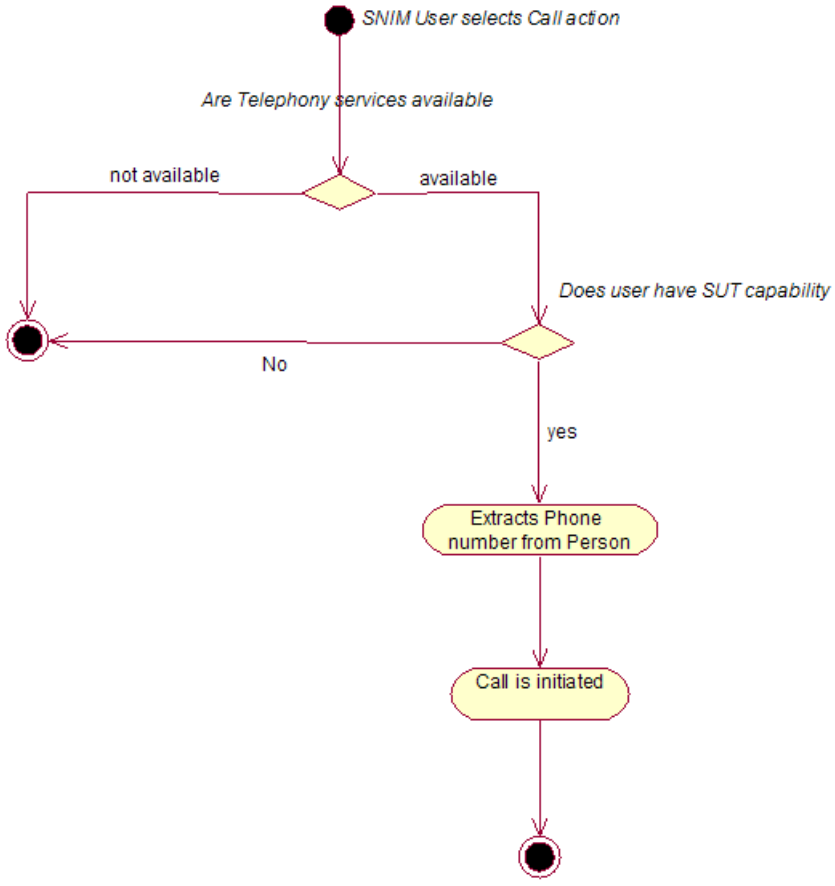


Figure 4.21: Activity diagram for "Call buddies" use case

Preconditions: The user must be logged into ST and also into the SN application.

Postconditions: Call will be initiated in case of no errors and in case of errors, ST notifies the user about the error occurred.

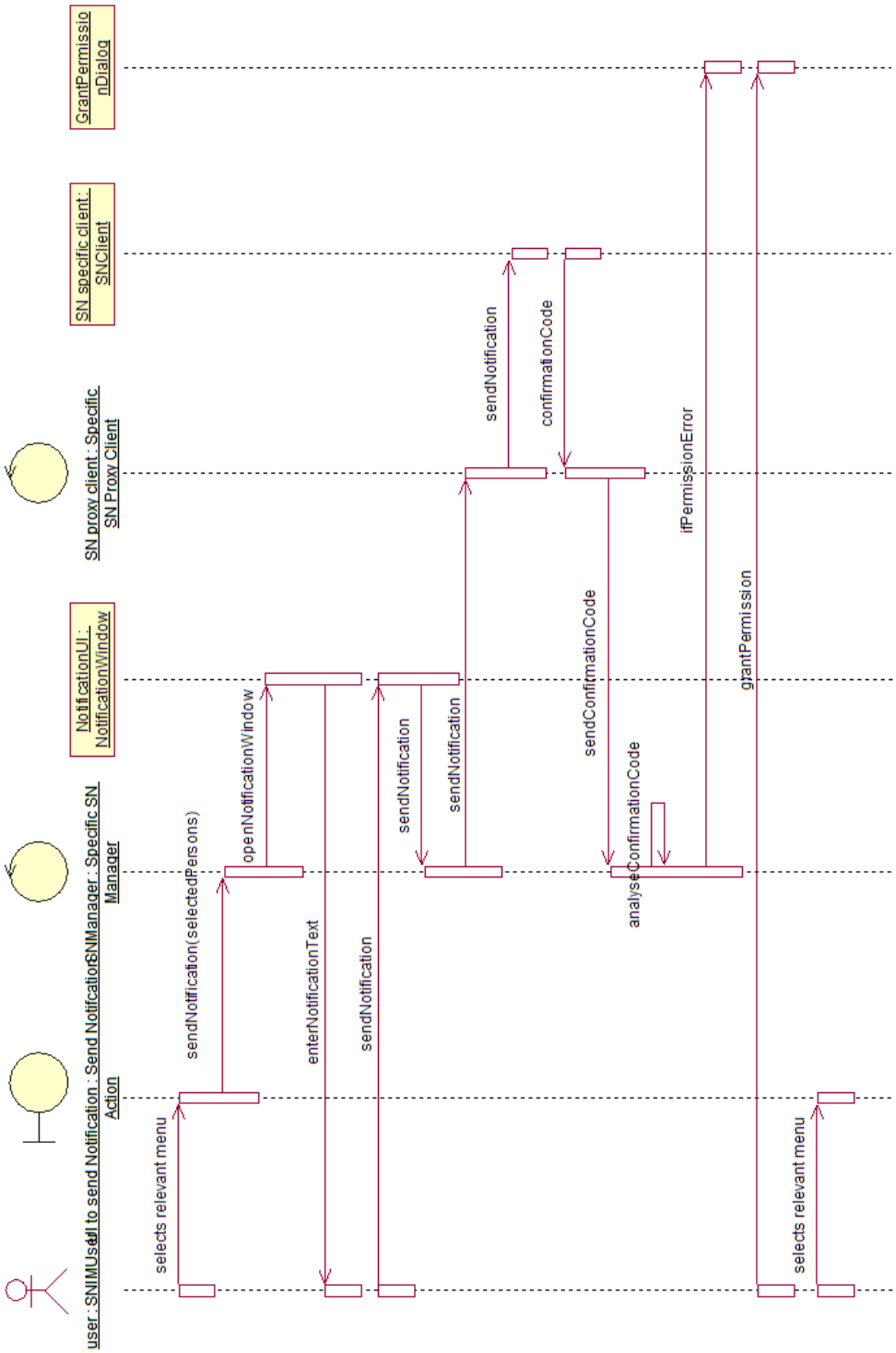


Figure 4.20: Sequence diagram for "Send Notifications" use case

4. **Use case - Add Friends to a group**

   Brief Description: This feature is part of the application and is used as part of the integration of IM and SN. This allows user to add friends of the SN user to a group of IM or ST in this case. This use case is initiated from the "Get Friends" interface from figure 4.10 and also as a startup task "Load friends and add to IM" shown in figure 4.19.

   Events:

   (a) User starts the action either by logging in or from the general actions UI.

   (b) The figure 4.22 depicts the events in this use case in form of a sequential diagram. This diagram shows the general action as the starting point of the use case.

   (c) The application responds to this user's action by running a thread that gets the friends of logged in user.

   (d) It interprets the response and returns the friends in form of SN users.

   (e) It creates the contacts in a format that IM understands and maps them to SN users.

   (f) It gets the preferences related to the group name to store the contacts that IM understands (referred as IM contacts or IM contact objects here onwards).

   (g) It adds the IM contacts which represent the friends of logged in SN user to the IM group.

   Preconditions: The user must be logged in to IM. The user should also authorise SN application but if has not already, then the application uses the login use case depicted in 4.18 to allow the user to authorise the application.

   Postconditions: The friends of logged in user are obtained and are added as IM contacts to a group.

4.4 Conclusion

The survey conducted in section 4.2.1 helped in framing the requirements of the software. The survey outcomings as discussed in the section 4.2.3 provided the starting point for the development of the software. It explained that majority of the corporate and home users would be interested in using an integrated solution for IM and SN systems as a desktop utility. The features that majority of users were interested in were also outlined in the section 4.2.3. Hence the analysis models(discussed in section 4.3) focused on those features for the first version of the software. They can be revisited for any improvements using another development cycle as outlined in the methodology(in section 3.2.4). The section 4.3 provided the business and system use cases and explained them using different UML diagrams. Now that the analysis of the project is done, the models developed in this phase can be used for design phase which is covered in the chapter 5.

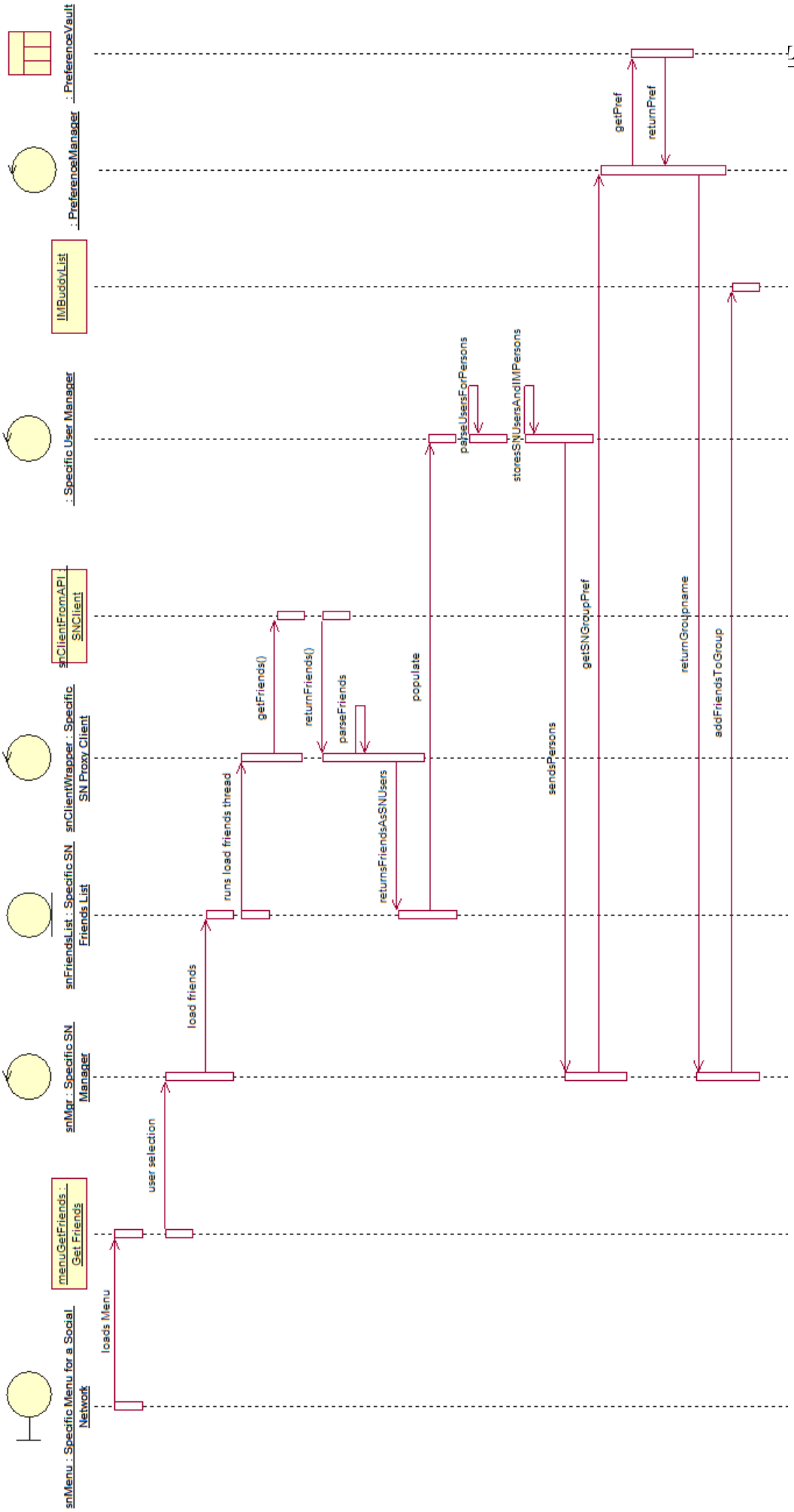


Figure 4.22: Sequence diagram for "Add friends to a group" use case

Chapter 5

Design

5.1 Introduction

The analysis models provided in chapter 4, explain the features and behaviour of the system. Having created the analysis models, the next phase would be to create design models which explain the structure and components of the system.

The software architecture in relation to the existing Sametime connect architecture will be explained in this chapter. The design models of the software based on the analysis models will also be explained. Along with the design modules, the design patterns used are also covered.

5.2 Architecture

The main architecture of the project is illustrated in figure 5.1. As the picture illustrates, the plugin(s) developed as part the project sits in the Sametime Connect client (the desktop client of IBM Lotus Sametime) and so it becomes part of the client, when installed, and communicates with Sametime server through the connect client or otherwise called as Sametime client or ST client in short. The plugin(s) that will be designed in this chapter are shown in the box named Sametime Social Architecture in the figure 5.1. The plugins and other components in this box are explained here in this section.

SN Integrator Plugin is the plugin that provides common functionality and interfaces required for each of the Social Network implementation modules represented by "SN Impl Plugin". It extends the extension points provided by Sametime for integrating with it and leaves the implementation to the implementation modules.

SN Impl Plugin SN Impl Plugin_1, SN Impl Plugin_2 and SN Impl Plugin_x in the figure 5.1 represent the implementation modules for social networks (SN_1, SN_2 ... SN_x) which contain the respective libraries (SN library_1, SN library_2 ... SN library_x - either readily available or as developed as required). The libraries connect and communicate with the respective Social network servers (SN Server_1, SN Server_2 ... SN Server_x) which process the requests made and supply the data requested. The implementation plugins use the framework laid out by the integrator plugin to provide the necessary data, related to the SN it represents, for integration.

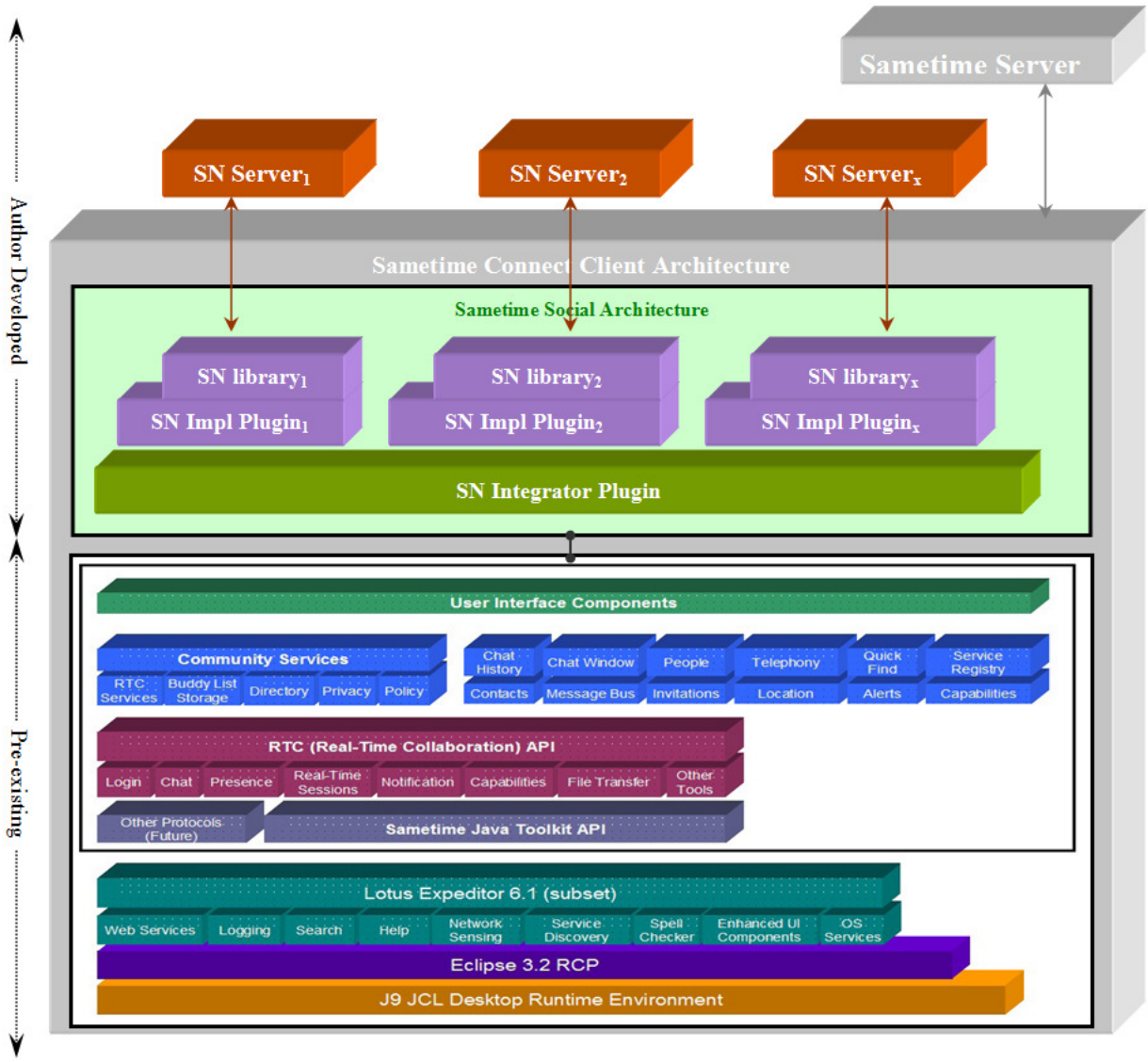


Figure 5.1: Architecture of the project - Sametime Social

5.3 Logical View - Design model

This section presents the design model of the project. This model in RUP is adapted to model the implementation environment. It thus serves as an abstraction of the source code. It also is considered as a "blueprint" for the structure and details of source code. It is a hierarchy of packages which inturn consists of classes. The classes on design model are abstractions of classes in the actual source code. The figure 5.2 shows the structure of the design model created. It shows the implementation packages for three SNs but similar is the structure for all SNs. This section explains the design for one of the implementation packages (Facebook implementation) and the same can be applied for the implementation package of any SN.

Sametime Connect Client: This package consists of the SDK for the Sametime client. The project uses this SDK for integrating SN into the ST.

Common API: This package relates to analysis model package "Integration module" which is shown in the figure 4.7. It consists of classes that are required for plugin architecture and the integration framework classes or interfaces that can be implemented for each SN to be integrated.

Facebook implementation: This package consists of classes needed for the integration

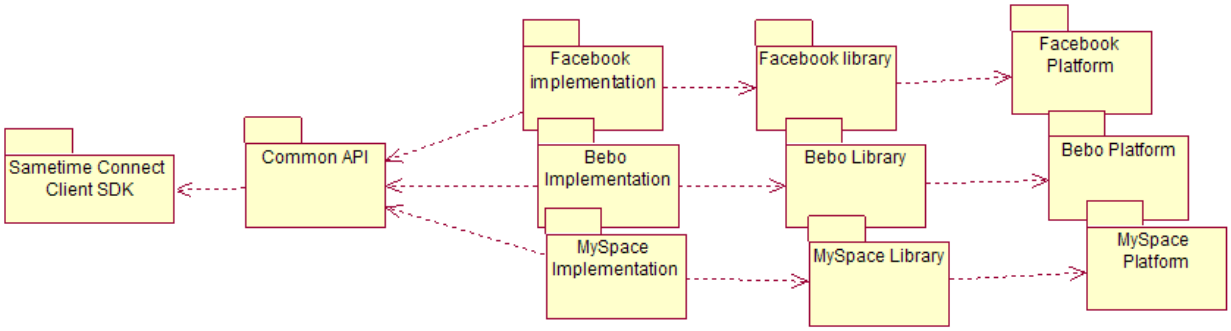


Figure 5.2: Architectural View

of a single SN with ST. It implements the framework laid out by Common API package and extends the basic functionality it provides. This package provides the support to the SN, for which it is developed, by running from the ST environment hooking up with ST client through Common API.

Facebook Library: This is the library required to integrate Facebook with ST. It used to be supplied by Facebook directly but they stopped supporting it in May, 2008 and suggested few alternative libraries. It is not part of the project but a library it needs.

Facebook Platform: This is not part of the project but just shown for clarity of design model structure.

5.3.1 Common API

This package consists of the API that implementation packages need to implement for integration of corresponding SNs. These classes are organised into four packages as shown in figure 5.3.

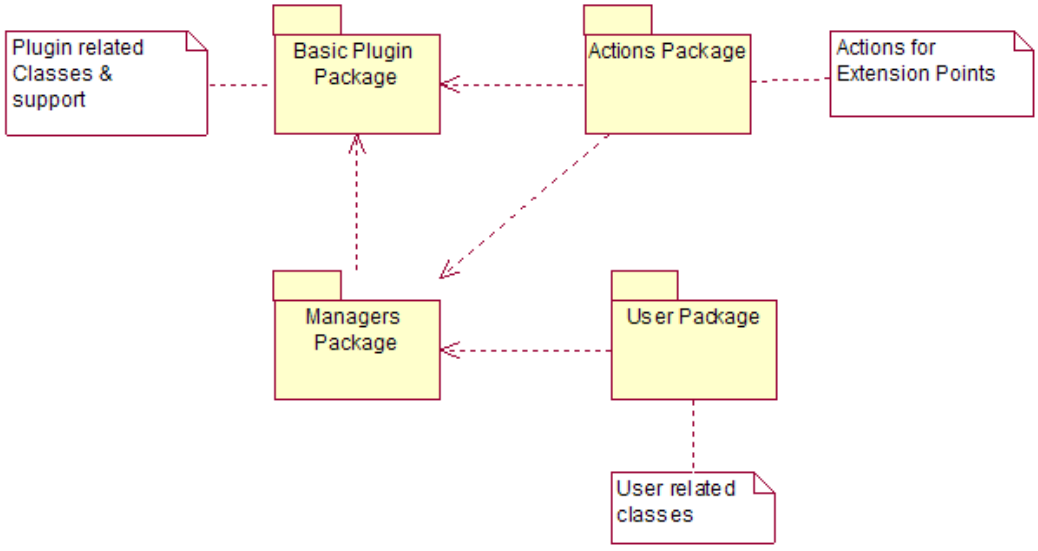


Figure 5.3: Package structure of Common API

1. **Basic Plugin Package:** This package consists of classes related to plugin life cycle as shown in figure 5.4

   AbstractUIPlugin is a class from Eclipse framework on which ST is developed. This is an abstract class when extended provides lifecycle methods for the plugin.

PluginActivator is a class designed to be the Activator for the plugin and it extends AbstractUIPlugin.

CommunityListener is an interface from Sametime SDK which listens to community events like community login event and community status event.

SNCommunityListener is an abstract class provided for implementation packages if they have any functionality to perform on community events.

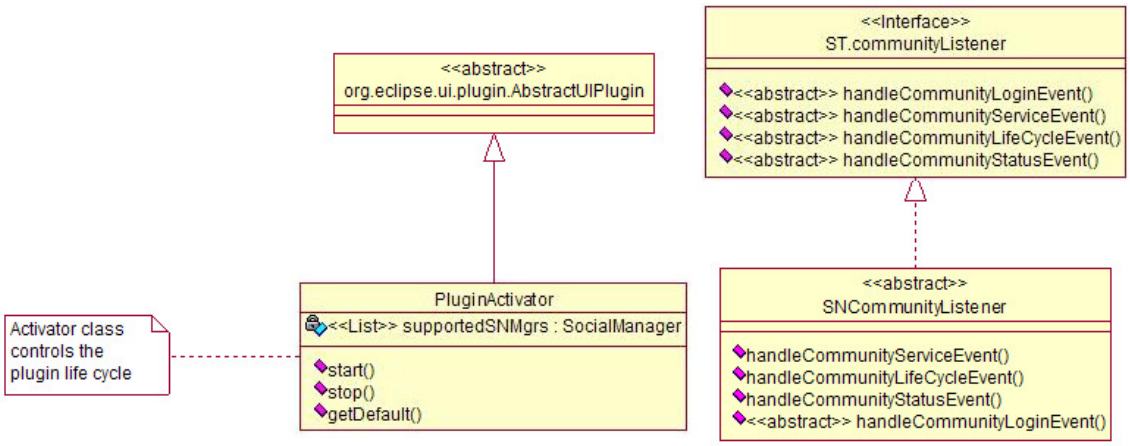


Figure 5.4: Classes in Basic Plugin Package

2. **Managers Package:** This package consists of classes that manage other classes. The figure 5.5 shows the classes that come into this package.

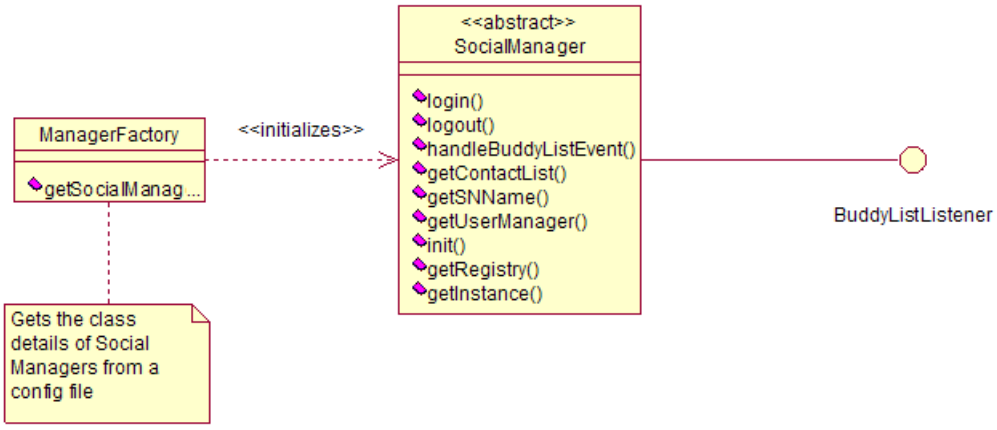


Figure 5.5: Classes in Managers Package

ManagerFactory is a factory class that instantiates manager classes for SNs. Section 5.4.2 explains the factory classes.

SocialManager is an abstract class which provides base for social networks to extend. It will be a singleton class when extended. It provides methods for accessing other classes which are part of implementation including User manager and Contact list and also contain the name of social network it manages. It also implements BuddyListListener from Sametime SDK. This listener listens to the buddy list events (Nielson, Juergensen, Menashes, Patton & Schejter 2002).

3. **User Package:** This package consists of classes that manage users. This package provides the interfaces required for the corresponding implementation classes. The figure shows the classes that belong to this package.

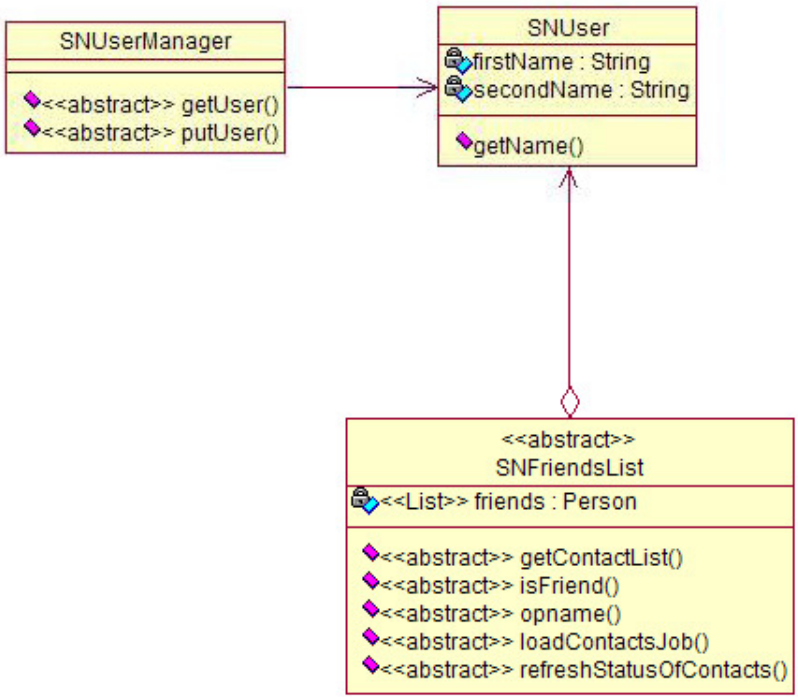


Figure 5.6: Classes in User Package

SNUser is the class that represents generic social network user. The class that extends it shall represent the user of specific SN that the class is packaged for. This class acts as a simple bean with fields or properties of person as it's attributes and the corresponding getters and setters as it's operations. The attributes of the class depend on the social network. The implementation class includes the attributes that the SN allows to use. Similar is the case with operations.

SNFriendsList is the abstract class that represents generic friends list. The class that extends it shall represent the friends list or contact list for the user in that SN that the class is packaged for. This class typically consists of the list of friends of the logged in user and some operations or methods that access this list, users in the list as well as the methods that load the users into the list, refreshing contacts. This class is an aggregate of SNUser class. The figure shows that it is an aggregate by reference because SNUser class is an abstract class and can not be instantiated as is.

4. **Actions Package:** This package consists of the classes responsible for UI actions. They associate themselves with the extension points provided by Eclipse framework and act as delegate classes for the UI actions. These classes are designed to provide a menu of list of SNs configured for integration where ever possible. When user selects a menu item, the action is delegated to corresponding SN's menu manager. Figure 5.7 shows the classes that belong to it.

StatusChangeAction is a class from ST SDK. This is an action class for actions that appear on the status toolbar in the ST's main window, above the buddy list, at the end

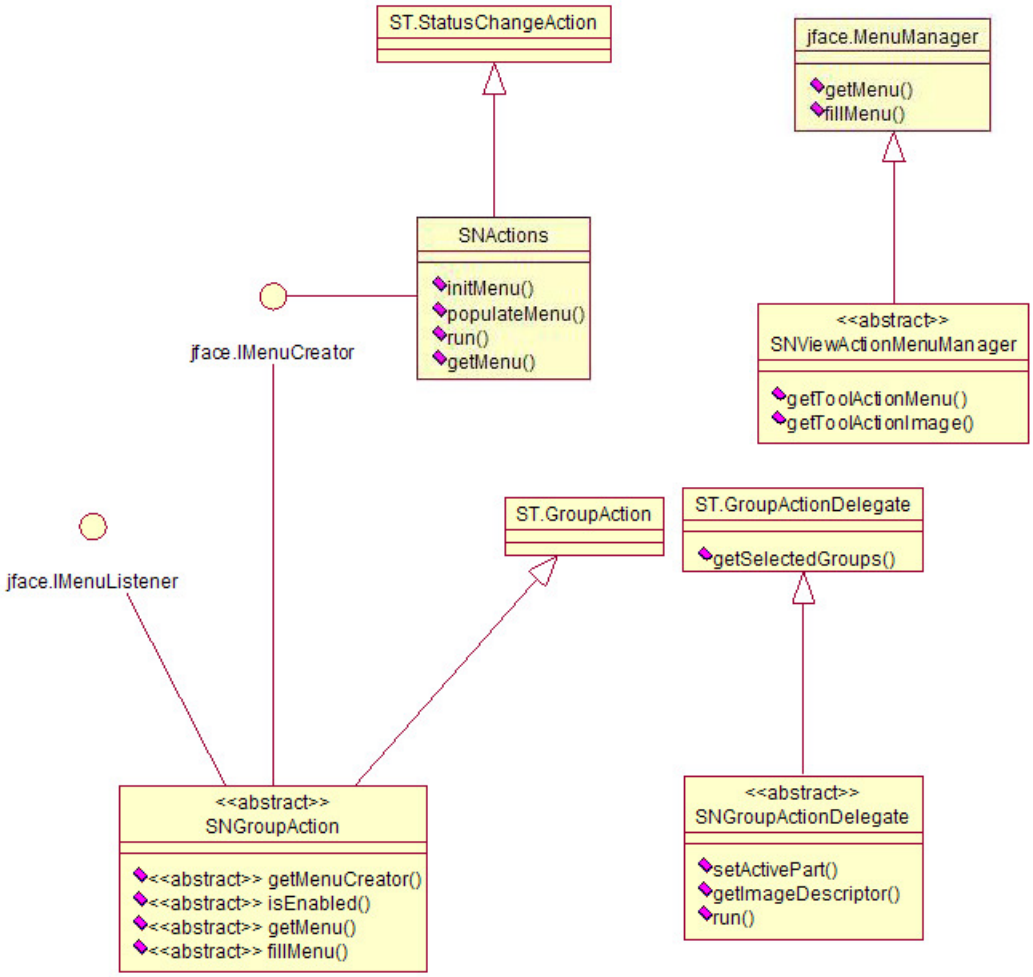


Figure 5.7: Classes in Actions Package

of current location as shown by red rectangle in figure 5.8.

SNActions extends "StatusChangeAction" and implements jface.IMenuCreator. "StatusChangeAction" delegates the action to this class. This class is designed to display a menu of a list of SNs configured for integration. It uses the factory method from "Managers Package" described in section 2 to get the list of SNs and then populates the menu. And the sub menus are populated using the implementation packages.

SNViewActionMenuManager is a menu manager template class. It provides methods for providing the menu items relevant to an SN which can not be determined at this level.

GroupActionDelegate is a class from ST SDK. It provides a means for providing extra functionality in the context menu of a group. When a user right clicks on a Group in buddy list, a pop-up menu appears and if an application wants to provide it's own menu item in that pop-up then, it needs to extend this class. It needs instances of "GroupAction" class to be added. It acts as a delegate to any actions added through the parent application and delegates the action to corresponding Group actions.

GroupAction is a class from ST SDK. It represents a Group action. This is the class that provides the actual action to the group's context menu.

SNGroupAction extends "GroupAction" to represent group actions for the current ap-

plication.

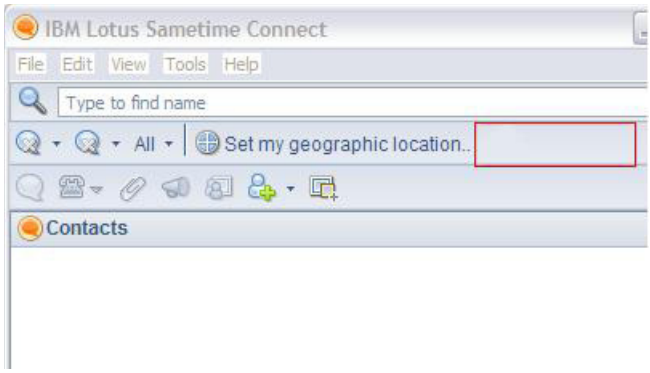


Figure 5.8: Sametime Connect - Status toolbar

5.3.2 Facebook Implementation

This package consists of classes that provide the integration of Facebook with ST. Majority of classes are designed to extend or implement the classes and interfaces in Common API package. The classes in this package are organised in different sub-packages each providing a specific type of function. The figure 5.9shows the structure of this package.

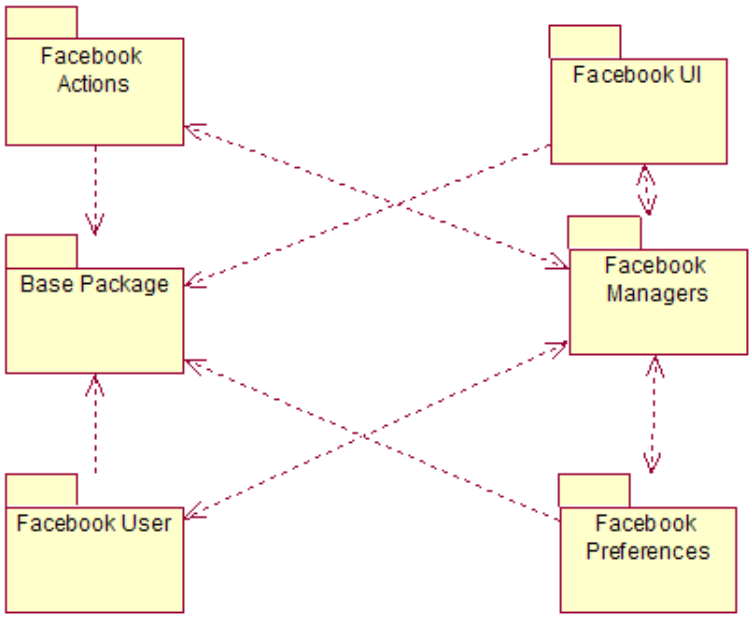


Figure 5.9: Structure of Facebook Implementation

1. **Facebook Managers:** This package consists of manager classes which implement the classes from section 2. The figure 5.10 shows the classes it contains.

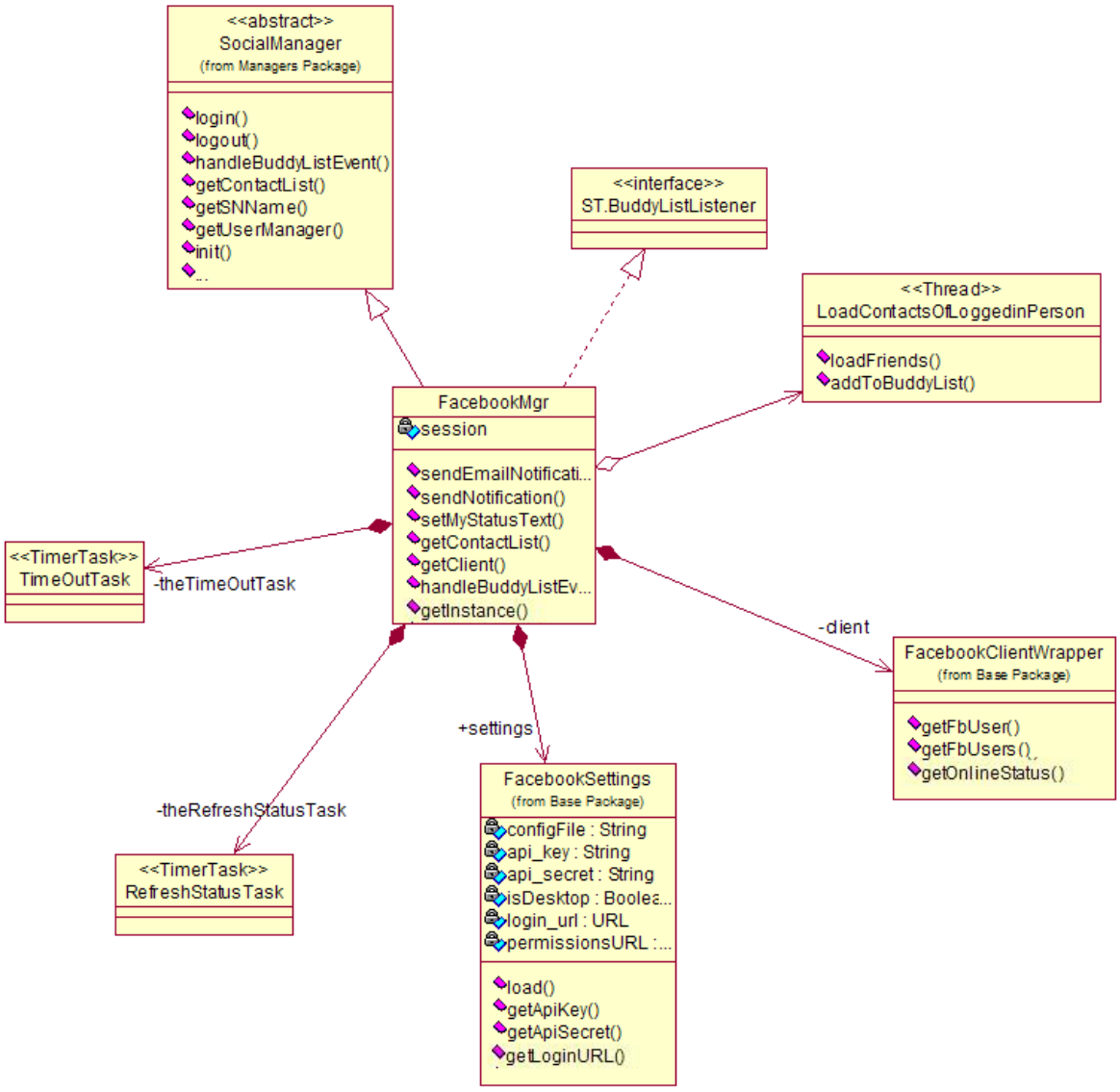


Figure 5.10: Classes in Facebook Managers package

FacebookMgr is a class that extends "SocialManager" class from Managers Package. It provides the methods related to Facebook features. It maintains the references to other manager classes, the client proxy class, settings class and some helper tasks which include RefreshStatusTask and TimeOutTask. It also implements the BuddyListListener for handling the buddy list events.

RefreshStatusTask is a timer task which is designed to execute periodically. This task, when run, makes a request to Facebook to return the status of the friends of the logged in user. This is needed because, this application being a pure desktop client application, there is no way for it to listen to any server events and hence the functionality is simulated using timer task.

TimeOutTask is a timer task designed to execute periodically as with RefreshStatusTask. The period is obtained from preferences. This task is used to simulate Facebook session time out.

LoadContactsOfLoggedInPerson is a thread that runs shortly after the user logs in.

It loads the friends of the logged in user by requesting Facebook, interpreting the results and creating SNUser objects. It then populates the SNUSerManager.The delay is used to avoid UI freeze soon after user is logged in and thus the loading process is done in the background.

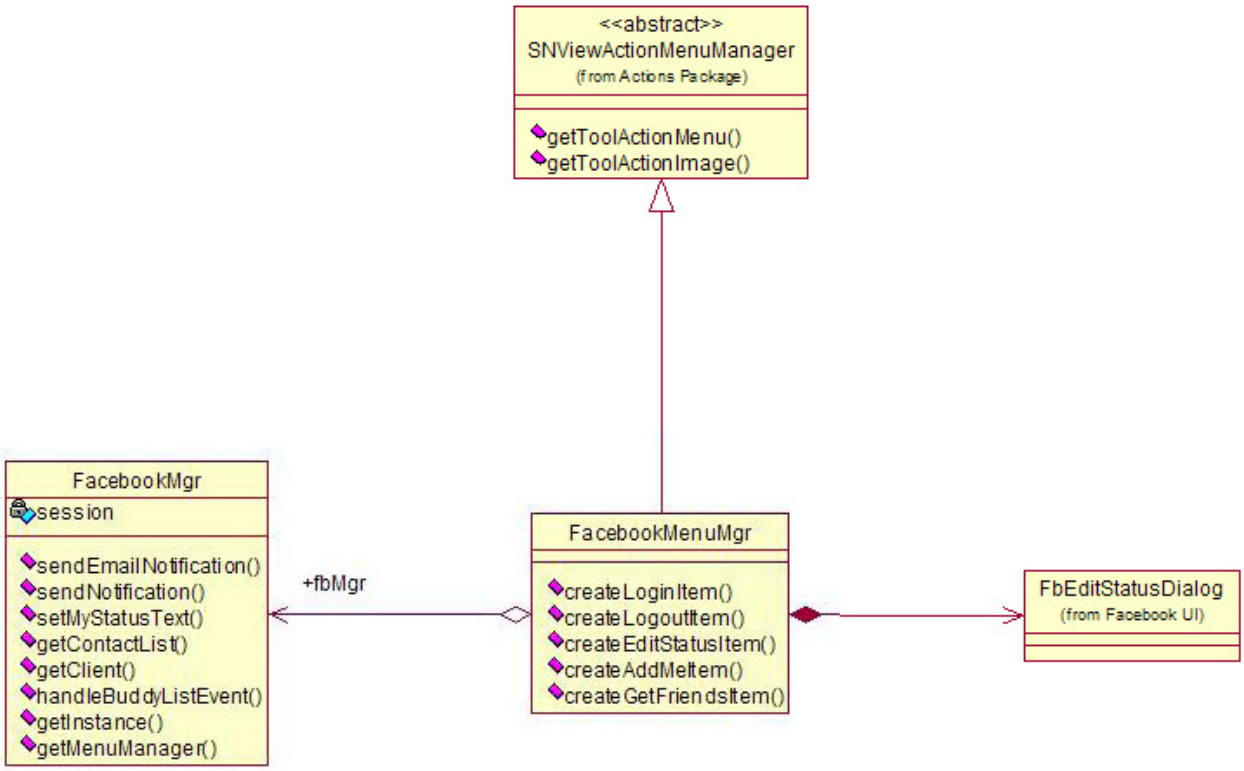


Figure 5.11: More classes in Facebook Managers Package

FacebookMenuMgr is a class that extends "SNViewActionMenuManager" from Common API package to provide common actions for Facebook including login, logout and change status.

2. **Facebook Actions:** This package consists of classes that are related to user actions. The classes extend or implement the action classes and interfaces designed in Common API's Actions package shown in figure 5.7. Figure shows the classes in this package.

   FbGroupMenuCreator is a class that creates the group actions. A group action is a menu item seen in the context menu of groups in the ST buddy list. The figure 5.13 shows an example Group action called "Add Contact" outlined in red for the group "Work". Similarly this class is designed to create group actions relevant to Facebook. The actions "SendNotificationEmailAction", "SendNotificationAction" and "SendLiveMessageAction" as shown in figure 5.12 are designed at this point. But for better accessibility, there will be one single group action for each SN and so in this package, the group action will be "Facebook" which will be a menu of actions each for supported feature.

   FbGroupAction is a class that is a group action itself. It extends SNGroupAction (an abstract class provided by Common API). It consists the menu creator which returns the group features provided by Facebook.

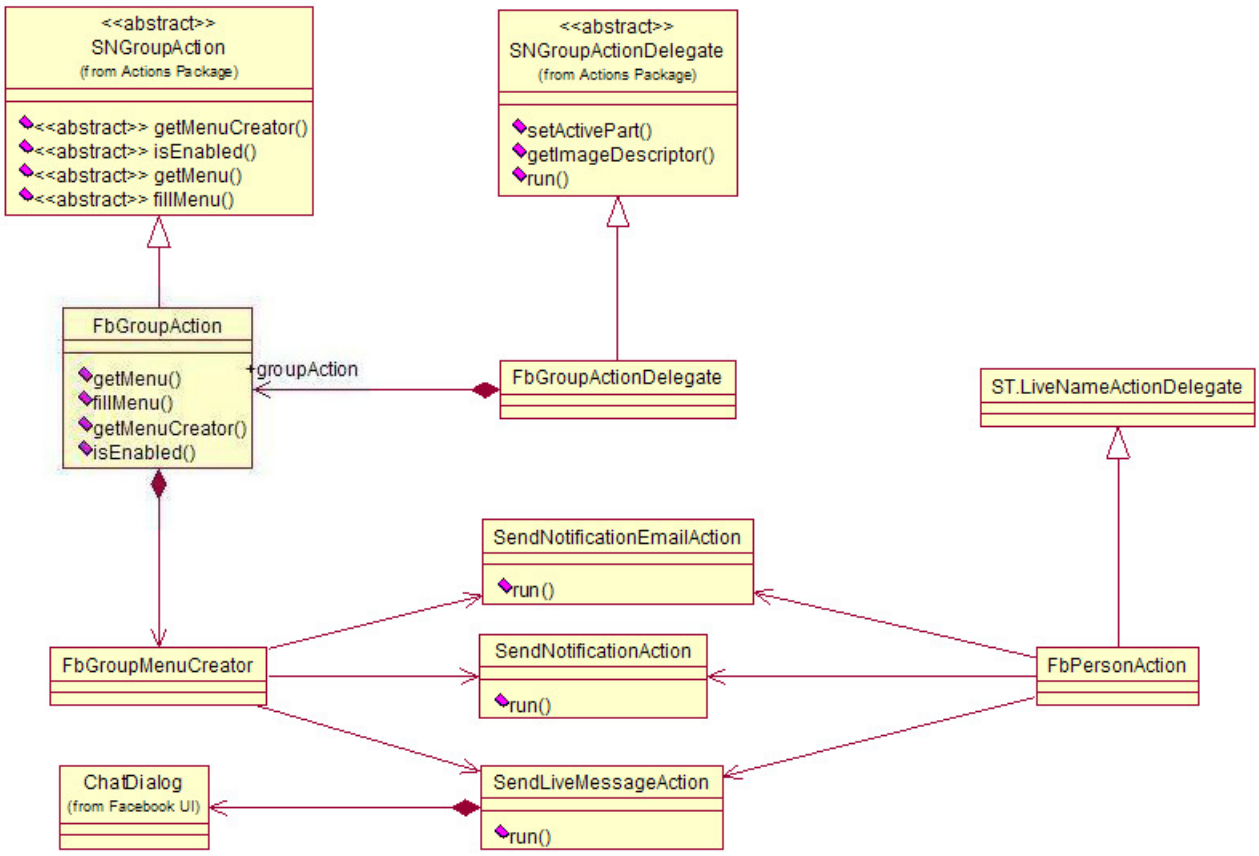


Figure 5.12: Classes in Facebook Actions package

FbGroupActionDelegate is a class that creates the group action and returns to the runtime while displaying the group context menu.

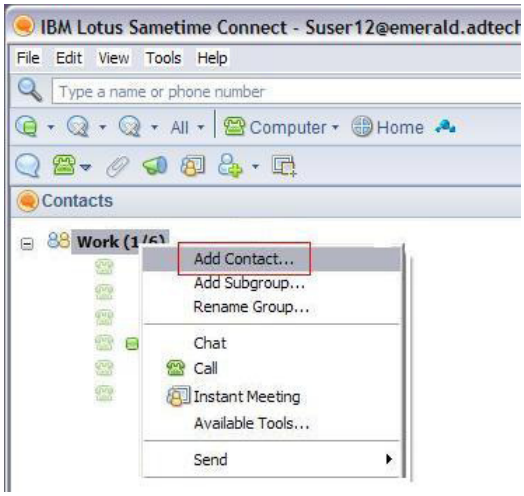


Figure 5.13: Example for ST Group Action

ST.LiveNameActionDelegate is a class that allows application to add actions to live names. Live names in ST are the contacts in buddy list and they are so called because of the inline status notification of contacts. Users can see the actions, added using this class, in the context menu of the contacts(live names or persons), in the buddy list. The figure

5.14 shows an example of person action or livename action called "Chat" outlined in red.

FbPersonAction is a class that extends the ST's person action interface "LiveNameActionDelegate" for providing person actions relevant to Facebook.

SendNotificationEmailAction is a class that represents both a person action and a group action, when run, sends the email notification to the person in case of person action and to the group in case of group action.

SendNotificationAction is a class that represents both person and group actions, when run, sends a notification to the person in case of person action and to the people in group in case of group action.

SendLiveMessageAction as with the actions above, this class, when run, sends a live message to the relevant people. But it needs the receiver(s) to be online.

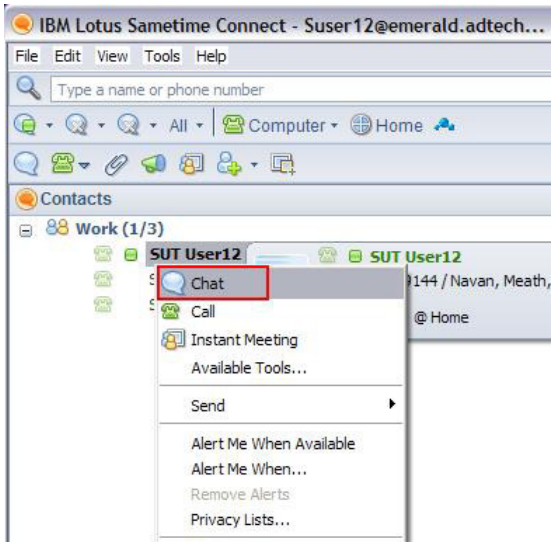


Figure 5.14: Example for ST Person Action

3. **Facebook UI Package:** This package consists of classes related to any custom UI created for Facebook. Figure 5.15 shows the classes of this package.

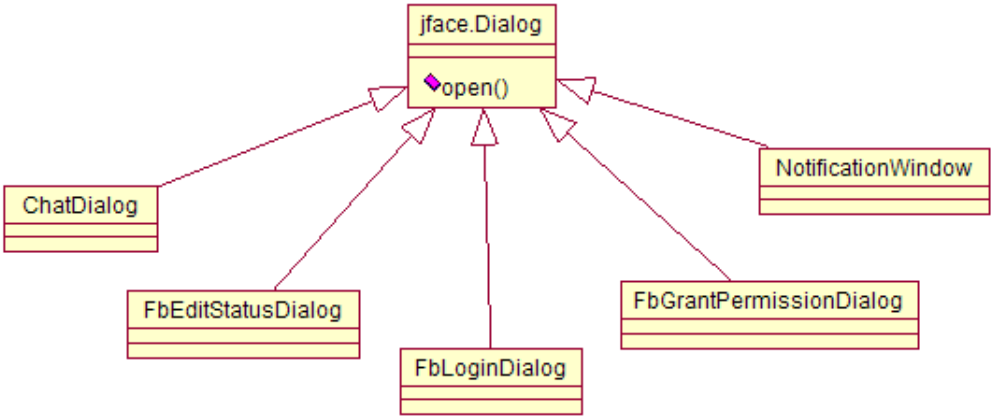


Figure 5.15: Classes in Facebook UI

All the UI designed at this stage are windows or dialog boxes for various purposes. All of these dialogs extend jface.Dialog from Eclipse framework.

ChatDialog is used to exchange live messages with online Facebook contacts.

FbEditStatusDialog is used to change the status text of the logged user.

FbLoginDialog is used for authorising the SN application using Facebook credentials as named as login process in the context of this application.

FbGrantPermissionDialog is used for granting any missing permissions. The SN application should have user permissions for certain tasks like email permission to send emails to a Facebook contact.

NotificationWindow is used to send notifications or notification emails to Facebook contacts.

4. **Base Package:** This package consists of classes that are used generally by all packages. Figure 5.16 shows the classes that come into this package.

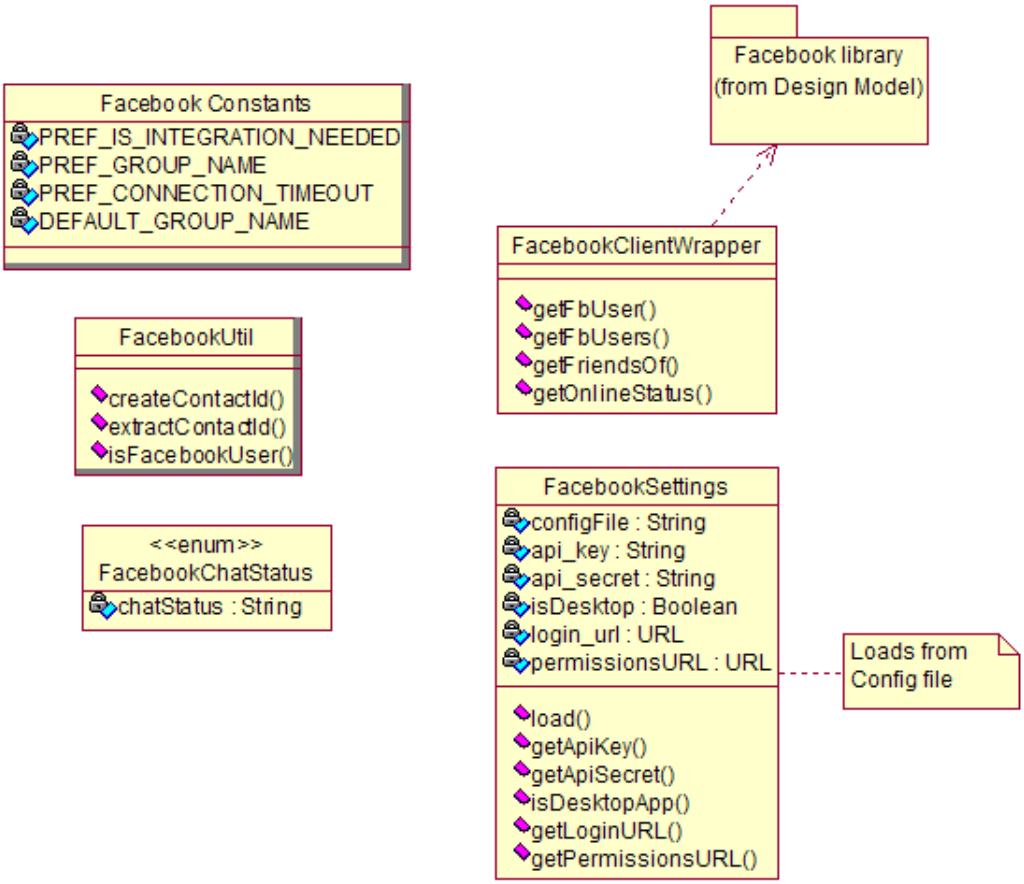


Figure 5.16: Classes in Base Package

FacebookConstants is designed as a placeholder for any constants used in the project. The constants shown in the figure 5.16 as attributes of this class are preference constants.

FacebookUtil is designed as the utility class for Facebook implementation package. It consists of commonly used methods in the project which might include creating recognisable contact ids for Facebook contacts. The word "recognisable" signifies that application should be able to identify the contacts using the contact id because there can be certain features which should only be available to Facebook contacts. The application should also be able to extract the id of the users that Facebook provides so that Facebook recognises them.

FacebookSettings is a class that wraps up all the parameters required for the integration. It loads the parameters from a file.

FacebookClientWrapper is a class that acts as proxy to the Facebook client from Facebook library. It delegates all the requests to the client from the API after wrapping all the parameters required for it to process the requests. It also helps in changing the formats of objects between Facebook and ST.

5. **Facebook User Package:** This package consists of the classes related to Facebook users (user objects). The figure 5.17 shows the classes that come under this package.

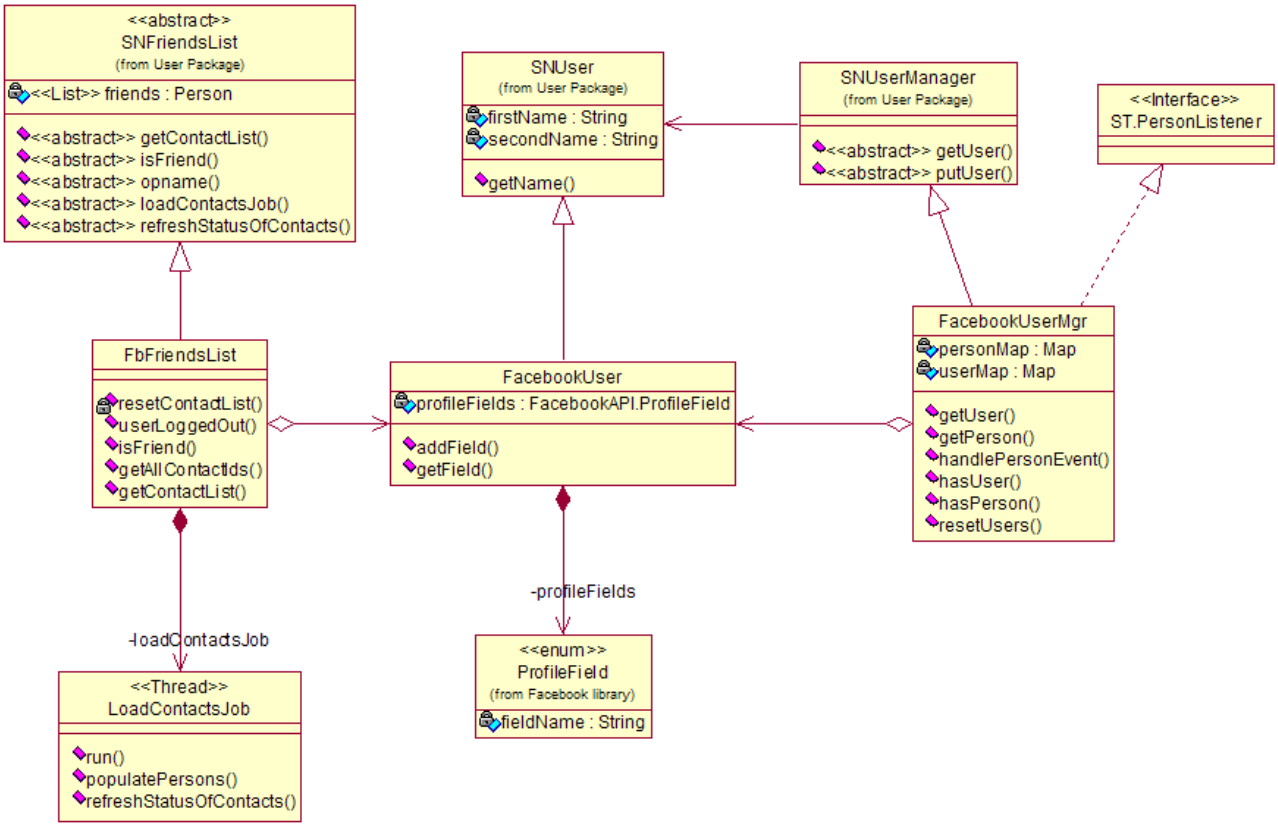


Figure 5.17: Classes in Facebook User Package

FacebookUser is a class that represents a Facebook user. It does not have any knowledge of ST. The attributes or properties of the Facebook user are maintained using an enum class called " ProfileField" which is provided by Facebook library.

FacebookUserMgr is a class that manages the Facebook user objects and the ST version of users called "Person" objects. It also maps between the two formats. It also implements "PersonListener" from ST to listen to Person events to maintain the consistent list of person objects and Facebook user objects in case of deletions of contacts in the buddy list.

FbFriendsList is a class that extends "SNFriendsList" class from "Common API" package. It contains the list of friends of logged in user as references to "SNUser" type objects. It acts as a simple bean for the friends list.

LoadContactsJob is a thread, when run, loads the friends of the logged in user and converts the Facebook users into ST persons. It also refreshes the status of friends.

6. **Facebook Preferences:** This package consists of classes related to preferences. It allows users to set some preferences for Facebook. The figure 5.18 shows the classes that come under this package.

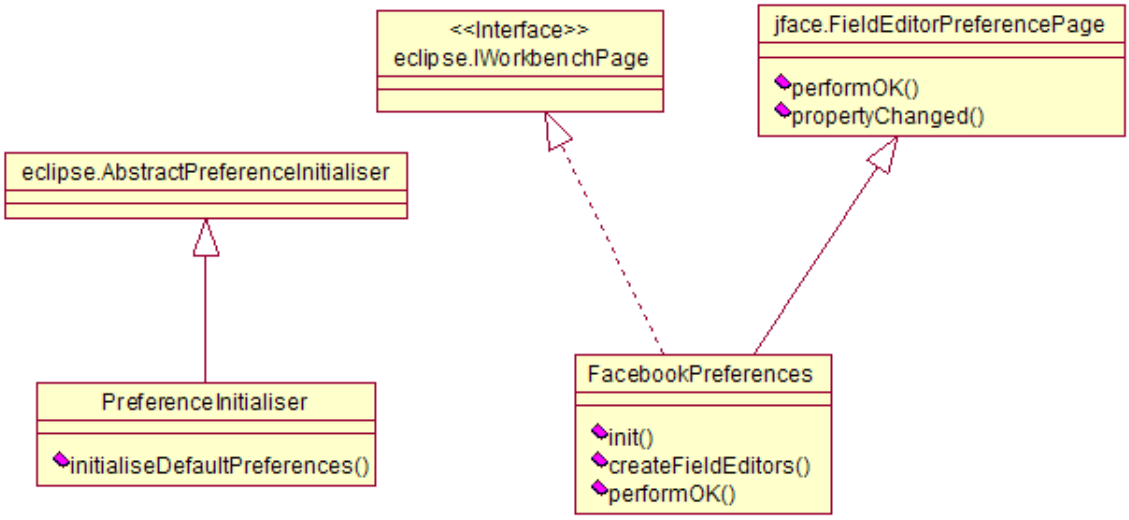


Figure 5.18: Classes in Facebook Preferences Package

FacebookPreferences is a class that creates a preference page for Facebook in ST preferences window. It extends "FieldEditorPreferencePage" from jface to manage any changed preferences and implements "IWorkbenchPage" from Eclipse to create a preference page.

PreferenceInitialiser is a class that extends "AbstractPreferenceInitialiser" (from Eclipse framework) to support preference initialisation. This class typically provides the default values for the preferences which are used when user does not set any preferences.

5.4 Design Patterns:

This section describes the design patterns used in this project directly. The software used while implementating this project might have many other design patterns followed but they are not covered here. For example, the programming languages might have used design patterns including Iterators and Composite and SN libraries might have used Proxy pattern for communication with the respective server.

5.4.1 Singleton

This is one of the commonly used creational patterns. Creational patterns are those which are used while creating objects in OO (Object Oriented) applications.

Brief Description: This design pattern ensures that a class has only one instance. Certain pieces of software need some classes to have exactly one instance like single printer spooler in a printer network, a single database connection in an application. Such classes are called singleton classes. A singleton class itself is responsible for not allowing more than one instance of itself to be created. It also provides a way to access the single instance. This is called Singleton pattern.

In this project, the Manager class shown in figure 5.10 is an example of Singleton class. Example code for a Singleton Class is in the figure

```
//Singleton Class
public class ConsoleLogger {

    private static ConsoleLogger cLogger;

    //Private constructor is used to avoid other
    //classes from instantiating this singleton class

    private ConsoleLogger(){
    }

    //public method to get the instance of this singleton
    public static ConsoleLogger getInstance(){
        if(cLogger == null){
            cLogger = new ConsoleLogger();
        }
        return cLogger;
    }

    //Other methods

    public void log(String msg){
        //Code to log the message
        System.out.println(msg);
    }
}
```

Figure 5.19: Singleton Pattern Example

5.4.2 Factory Method

Factory Method is another creational design pattern.

Brief Description: In some scenarios, an object might not know which class of a known class hierarchy it should instantiate. It might only know the class hierarchy but not the exact subclass among a set of subclasses of the common parent class. The reason might vary from situation to situation. Some times the decision might depend on the configuration settings or sometimes the state of application. In such cases, this design pattern is used. It recommends encapsulating the selection criteria to be implemented in a designated method and calls it the factory method. Thus the factory method selects an appropriate class from a known class hierarchy based on the implemented selection criteria, instantiates it and returns the instance of the parent class type. As it returns the object, of required class as parent class type, the application object, which calls this method for the instance, does not need to know about the exact class in class hierarchy. In this application, the ManagerFactory class shown in figure 5.5 is used to create SocialManagers based on the configuration file. A simple implementation of this factory pattern decides whether to use local file logger or send the log messages to a remote system based on a configuration file. The figures 5.20 and 5.21 show a simple example of how this can be implemented.

```
//Superclass
public class Logger {

    public void log(String msg){
        //Any default implementation like System.Out
    }
}


// A subclass for Logger
public class FileLogger extends Logger {

    public void log (String msg){
        //file logger implementation
        // To print log message to a file
    }

}

//Another subclass of Logger
public class RemoteLogger extends Logger {

    public void log(String msg){
        //Implementation of remote logging.
    }
}

//Class that uses factory method to get the appropriate logger instance.
//It need not know which type of logger to use.

public class ExampleFactoryMethod {

    public static void main(String[] args) {
        LoggerFactory lf = new LoggerFactory();
        Logger _logger = lf.getLogger();
        _logger.log("Test log message");
    }

}
```

Figure 5.20: Class Hierarchy that factory method knows

```
public class LoggerFactory {

    //Read configuration file
    public String getLoggerTypeToUse(){
        //Create Properties object to hold the configuration file's contents
        Properties prop = new Properties();
        try {
            prop.load(ClassLoader.getSystemResourceAsStream("logger.conf"));
            return prop.getProperty("Logger_to_use");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;

    }

    //Factory method that implements the selection logic and
    //returns the instantiated object at runtime


    public Logger getLogger(){
        String loggerType = getLoggerTypeToUse();
        if(loggerType!=null && loggerType.equalsIgnoreCase("remote")){
            return new RemoteLogger();
        }
        return new FileLogger();
    }

}
//Class that uses factory method to get the appropriate logger instance.
// It need not know which type of logger to use.
public class ExampleClass {

    public static void main(String[] args) {
        LoggerFactory lf = new LoggerFactory();
        Logger _logger = lf.getLogger();
        _logger.log("Test log message");
    }

}
```

Figure 5.21: Sample code for Factory Method and an application object that uses it

5.5 Conclusion

The architecture of the project was explained in relation to the Sametime connect architecture. Design models of this project were created based on the analysis models from chapter 4 and the architecture explained in section 5.1. The models were UML diagrams created using Rational Rose and most of them were class diagrams. This chapter also presented the brief idea of where would the UI, developed as part of the project, be seen in the Sametime Connect client. The what and why of the design patterns used were also explained along with sample code. The models created in this chapter would be developed to create a working version of the product using appropriate software and this is covered in the chapter 6.

Chapter 6

Implementation

6.1 Introduction

The design models developed in the chapter 5 were used in the implementation phase of the software, to develop the classes required. Hence the implemented software could be organised based on the modules created in the design phase. Implementation phase now should comprise of activities including selecting appropriate softwares and implementing the design models to form the workable software. These activities are covered in this chapter along with any issues faced during implementation and how they were overcome. The result of this phase would be a testable software. This phase was done in iterations each iteration to implement one use case. This phase again can be repeated for each SN that has to be integrated.

6.2 Choice of Softwares

This section presents some of the decisions made for choosing a social network to integrate with IBM Sametime.

6.2.1 Choice of IM

The choice of IM is already made as mentioned in chapter 1 to use IBM Lotus Sametime. The choice is also supported by some points mentioned in the listing below

1. IBM Lotus Sametime provides extensible development environment by making use of Eclipse framework. This framework is explained in detail in section 6.3.
2. It is an award winning solution (IBM Lotus Sametime n.d.) for communication and collaboration.
3. It is an enterprise solution which also allows the public IMs through Gateways.
4. It allows telephoning the contacts added.
5. It provides rich user experience.
6. It provides good security.

6.2.2 Choice of Social Network

There are many social networks to choose from. The idea is to provide integration to any social network regardless of what type of SDK they provide. The design modelled in chapter 5 is generic for all social networks and the implementation differs based on the technology of the SDK the corresponding server supports. Thus the technical details are left to the implementation modules. So the implementation is started with one SN and the implementation for other SNs will be planned as time permits. Adding an implementation module for another SN can be done as a new iteration or completely as a new version for the software.

Many social networks do not provide any information about the development support. It was not a straightforward research to find out about the development support in case of many SNs except a few including Facebook, Bebo, Myspace, Friendster and Orkut. The following listing briefly presents some research done while choosing initial SN for implementation.

6.2.2.1 Facebook

1. The initial motivation was the small questionnaire, filled by a some SN users, in the Requirements phase outlined in section 4.2.1. It showed that most of the corporate and home users use Facebook.

2. Facebook also claims that there are about 120 million active users (Facebook Press Room n.d.) as on November 2008.

3. It provides development support in wide range of technologies including REST (**RE**presentational **S**tate **T**ransfer), PHP, Python, FBML(**F**acebook **M**arkup **L**anguage), FQL(**F**acebook **Q**uery **L**anguage) and Ruby. REST is helpful to use in desktop application.

4. There are many open source libraries available though Facebook has stopped official support for Java, so there would be no need to implement a library anew.

6.2.2.2 Bebo

1. Bebo provides development support in Java, PHP and Ruby. Java can be used in desktop applications.

2. There is no official information on how many active users are there.

3. Some of the API only assume the users who are members of the application being implemented while processing the requests. For example API related to getting friends of a user returns only the friends who are also the members of the current application (which sends the request).

6.2.2.3 Orkut

It provides development support only in OpenSocial 0.7. OpenSocial 0.7 only supports online applications (either run on OpenSocial's sandbox or a custom one). So it does not suit the development of desktop application. OpenSocial 0.8 supports desktop applications and the API of this version is not yet available .

6.2.2.4 Hi5

It provides OpenSocial 0.8 support which allows REST API and thus supports the development of desktop applications. But, the API is still in Beta as of September (Hi5 2008). It provides a separate Sandbox, betasandbox.hi5.com, to run Hi5 applications .

6.2.2.5 Plaxo, MySpace

They provide REST API but there is no official library. Hence a custom client library has to be implemented.

So as a proof of concept for the integration of social networks through a common API, Facebook is chosen as a starting point to integrate with Sametime. Rest of this chapter presents a walk-through of the development stage of the project.

6.2.3 Software

The softwares used for development and testing of this project are outlined in the table 6.1. It lists the type of software used, it's name and version and brief reason why used it and how.

Table 6.1: Softwares used in the project

Software type	**Software name**	**Software Version**	**Why and Where**
Operating System	Windows XP	SP2	The system used to develop the software and test it
Operating System	Windows 2000	SP4	Used only for testing
Operating System	Ubuntu (Linux)	8.0.4	Used only for testing
IDE	Eclipse SDK	3.2.2	For development of plugins (both integrator and implementation)
Java Compiler	Java SDK	5.0	Used for development and testing
Java Runtime	J9 JCL Desktop	6.2	Runtime recommended by Sametime SDK
Product Source	Sametime Internal API	8.0.1	Used for developing a sametime application
Product	Sametime Connect Client	8.0.1	Used to run the application
Library	Facebook Client library	1.8.0 beta and final	Used to develop Facebook implementation module

Latest Facebook library available at the time of implementation was 1.8.0 beta but by the time implementation finished it was 1.8.0 final. So the application was verified with both the versions. At the time of documentation of the project, the latest version was 2.0.3 with which the application was not tested. 1.8.0 final version of the library is available at Google code website (Google Code 2008).

6.3 Plugins and Extension points

Eclipse platform is a framework used to build any sort of IDE. This is the framework used by Sametime Connect client. So the same framework is used for the development of the project, Sametime Social. Eclipse IDE is built based on plugin architecture. A plugin, in Eclipse, is a basic unit of function or component. The Eclipse platform itself is composed of plugins and so are the tools that extend it. A plugin is packaged with everything required to run the component for which it is built. It also consists of a manifest file called *plugin.xml*. The details including

interconnections with other plugins, it's dependencies and visibility of it's classes to other plugins are contained in this file.

A plugin will be able to expose parts of it's functionality to other plugins, as well as, use functionality exposed by other plugins. These requirements will also be declared in the manifest file. The declarations pertaining to the functionality exposed to other plugins are called "Extension Points" and the declarations pertaining to using the functionality exposed by other plugins are called "Extensions". When the Eclipse application is run, all the available plugins are discovered by the platform and the declarations in the manifest files are processed and the plugins are linked together. A plugin, by default, is activated when the code in it has to be run; but this can be configured to be activated at the time of application launch. A plugin, when activated, will be provided it's own class loader. This class loader enforces the visibility, of the classes and functionality, as declared in the manifest file.

6.4 Integrator Module

This is the plugin that integrates social networks with Sametime connect client and thus provides users a single interface to access both Instant messaging and Social Network systems. It provides the common framework for the integration which is then used and implemented by the implementation modules. This section explains how this plugin works. This plugin is spread across different packages depending on the type of functionality it provides.

6.4.1 com.dit.st.sn.integration

This package contains the classes that are required as a plugin and required to be at the plugin level.

Activator is a class that every plugin must contain and it controls the life cycle of the plugin. On start of the plugin (when plugin is loaded by the runtime), it initiates all social managers. But the list of social networks to initialise is not known at this time and so a separate ManagerFactory is used which returns the social managers for the SNs that need integration.

SNCommunityListener is an abstract class that provides a way for the implementation modules to listen to the community events including CommunityLifeCycleEvent, CommunityLoginEvent, CommunityServiceEvent and CommunityStatusEvent. These events are explained in detail in the toolkit (Ott 2008)

6.4.2 com.dit.st.sn.integration.manager

This package contains the classes that manage other parts of the plugin and are explained in this section.

ManagerFactory is a class that manages the instances of the social managers that are configured for integration. Each social manager manages the requests, responses and supporting objects related to one SN. ManagerFactory reads the list of social managers and the location of manager classes from a configuration file. The file (SNList.conf) used in the project contains the entries for Facebook and Bebo as shown in the figure 6.1.

After reading from the file, it instantiates each of the social managers using reflection and maintains them. The code that does this work is shown in the figure 6.2

```
Contents of a configuration file that ManagerFactory class loads
# List of Social Network managers ready for integration
# Format used is <SN name> = <SN Manager class location>
Facebook = com.dit.st.sn.integration.Facebook.impl.FacebookManager
Bebo = com.dit.st.sn.integration.Bebo.impl.BeboManager
```

Figure 6.1: Sample contents of SNList.conf used by ManagerFactory

```
snMgrClassNames = loadMgrClassNames();
Set set = snMgrClassNames.entrySet();
Iterator i = set.iterator();
while(i.hasNext()){
    Map.Entry entry = (Map.Entry)i.next();
    String socialManagerClassName =(String) entry.getValue();
    Class cl = Class.forName(socialManagerClassName);
    java.lang.reflect.Constructor constr = cl.getConstructor(clParam);
    snMgr = (SocialManager) constr.newInstance(objParam);
    snMgrs.put(snMgr.getSNName(),snMgr);
}
```

Figure 6.2: Code that reads the SNList.conf file and instantiates the SN managers

6.4.3 com.dit.st.sn.actions

This package contains the classes that extend the extension points of sametime connect client. These classes are later used by implementation modules.

SNActions extends the class called StatusChangeAction which allows it to display a UI component at the location shown in figure 5.8. **StatusChangeAction** is available from the toolkit. This class also creates a menu with a submenu for each of the SN. The submenus are created by the SN implementation modules. For the menu to appear at the status tool bar, the plugin has to contribute to the extension point, org.eclipse.ui.viewActions, as shown in the figure 6.3.

```
<extension
      point="org.eclipse.ui.viewActions">
        <viewContribution
         id="com.dit.st.sn.integration.viewAction"
         targetID="com.ibm.collaboration.realtime.imhub">
      <action
            class="com.dit.st.sn.integration.actions.SNActions"
            icon="images/My-friends-blue.gif"
            id="com.dit.st.sn.integration.actions.SNActions"
            style="push"
     tooltip="Common actions for Social Networks"/>
            />
   </viewContribution>
</extension>
```

Figure 6.3: Code that contributes to extension point, org.eclipse.ui.viewActions

It shows the menu as an icon(the image - images/My-friends-blue.gif , the relevant code is shown in figure 6.3). It generates the submenus at runtime depending on the available SN implementation modules. This code is shown in the figure 6.4.

```
Collection<SocialManager> snMgrs = Activator.getDefault().getLoadedSNMgrs();
if ( snMgrs!=null && snMgrs.size()>0){
    snItr = snMgrs.iterator();
    while(snItr.hasNext()){
        snMgr = (SocialManager)snItr.next();
        if(snMgr!=null){
            mi = new MenuItem(menu2,SWT.CASCADE);
            mi.setText(snMgr.getSNName());
            mi.setMenu(snMgr.getMenuMgr().getToolActionMenu(mi));
            mi.setImage(snMgr.getMenuMgr().getToolActionImage());
        }
    }
}
```

Figure 6.4: Code that populates menu in the status toolbar

SNGroupActionDelegate extends the ST class GroupActionDelegate in order to add a new group menu to the group's context menu in ST buddylist. For this reason, it contributes to the extension point called org.eclipse.ui.popupMenus and to the object, com.ibm.collaboration.realtime.livenames.GroupSelection, as shown in figure 6.5. The application after contributing to the relevant extension point, should also be able to handle the group events (like events generated when user selects the group menus). For this, it should provide a delegate class that extends com.ibm.collaboration.realtime.people.internal.livenames.GroupActionDelegate. SNGroupActionDelegate serves this purpose and handles the group events. This class populates the menu whose menu items are the SNs configured for integration. The code that provides this functionality is similar to code shown in figure 6.4. Each such menu item is in turn a menu for the group actions available for that particular SN and is implemented in class, SNGroupAction .

```
<extension
       point="org.eclipse.ui.popupMenus">
  <objectContribution
         adaptable="false"
         id="com.dit.st.sn.integration.actions.SNGroupContribution"
         objectClass="com.ibm.collaboration.realtime.livenames.GroupSelection">
      <action
            class="com.dit.st.sn.integration.actions.SNGroupActionDelegate"
            id="com.dit.st.sn.integration.actions.SNGroupActionDelegate"
            label="SN Group Actions"
            menubarPath="additions"
            style = "pulldown"/>
    </objectContribution>
 </extension>
```

Figure 6.5: Code that contributes to extension point for adding actions to Groups

SNGroupAction creates a group action menu which is relevant to a particular SN. Such menu will be displayed inside the generic menu created by SNGroupActionDelegate. This ab-

stract class is to be implemented by the implementation modules for adding relevant group actions.

6.5 Implementation Module for Facebook

com.dit.st.sn.integration.facebook.*

These packages consists of classes related to Facebook implementation. These packages collectively form an implementation module for Facebook. This module implements the integration module discussed thus far in this chapter. The implementation modules for different Social networks are pluggable. Addition or removal of each implementation module just adds or removes the integration of the SN, it represents, with ST. This section shows the important parts of Facebook implementation. For Facebook implementation, this project uses REST API provided by Facebook platform because this project is designed to be pure client implementation.

Configuring an implementation module involves only two steps enlisted below :

- Implement the module using the framework laid down by integrator module and based on the technology planned to use for communicating with SN server.
- Provide the location of SocialManager to the integrator module in the file SNList.conf as shown in figure 6.1.

FacebookManager is a class that manages the functionality and coordinates the objects of the Facebook implementation (of the integrator module). When plugin is loaded, the integrator module instantiates all the configured SNs. In initiation phase, a SocialManager loads the FacebookSettings which contains the information related to the Facebook application created at the Facebook website. The information required for creating a proxy client to make further Facebook requests on behalf of the Facebook application are populated in the settings.conf file. The mandatory properties are shown in figure 6.6. Then it initialises FacebookMenuManager which displays the Facebook related common actions in the ST status toolbar and FacebookUserManager which maintains the Facebook users and corresponding ST objects . It also listens to the *BuddyListListener* to handle buddy list events for example, on expanding a collapsed group (which causes a buddy list event), refreshes the status of the group members.

```
api_key          = [redacted]
secret           = [redacted]
login_url        = http://www.facebook.com/login.php?v=1.0&skipcookie=1
grant_permission_url = http://www.facebook.com/authorize.php?v=1.0
desktop          = 1
```

Figure 6.6: Sample Settings.conf for Facebook

On login action, this class starts three tasks to simulate client-server event handling mechanism. After a user logs into the Facebook application, Facebook manager starts an asynchronous task that loads all the friends of the logged in user. It is implemented asynchronously to avoid any delays caused by the requests and responses involved in getting the friends list. Batch API is used to get the user information of the friends list to increase the processing speed. Second task is a timer task which runs periodically given the period.

This task simulates session timing out. Users set the session timeout period and it is sent to Facebook as session timeout parameter while logging in. So the timer task informs the users that their session has timed out at the server. Users will be transparent to this task and tries to relogin. Third task is also a timer task whose period is again implemented as a user preference and this task refreshes the status of all the friends of the user. The general implementation in case of non desktop applications would be that the client registers with relevant events and will get the user status change events from the server. This task simulates this functionality except that it is done periodically but not as soon as the status change happens. In case of smaller refresh periods, the lapse between actual status change and when it is visible in ST narrows down but at the cost of more traffic. The code that does this is shown in the figure 6.7 . All these tasks are stopped when user logs out of Facebook as well as ST.

```
// Refresh status of the contacts regularly
TimerTask tt = new TimerTask() {

    @Override
    public void run() {
        ContactList cl = getContactList();
        if (cl != null) {
            cl.refreshStatusOfContacts();
        }

    }

};
if (refreshStatusTimerjob == null) {
    refreshStatusTimerjob = new Timer();
} // 3 minutes in milliseconds
refreshStatusTimerjob.schedule(tt, new Date(), 3 * 60 * 1000);
```

Figure 6.7: Timer task that refreshes status of Facebook users

FacebookManager also maintains the list of user properties to retrieve from Facebook. In Facebook terminology, the user properties are called ProfileFields. At the time of creation of user's friends list, it gets the user fields from Facebook and creates Facebook users and then create ST users and stores them in user manager. This is done using the code shown in figure 6.8.

FacebookUserManager manages all the user related objects in both Facebook format and ST format and provides mapping between them. It supplies the users in either Facebook format or ST format as requested and caches the logged in user. It also clears all the users on receiving user log out event or community log out event and then gets populated when the user logs in again. It does so to facilitate users to login with different Facebook accounts or login from a different client, which is not an uncommon scenario. Doing so will also make the application analogous to Sametime connect client. It makes the list consistent to what is displayed in the client by listening to PersonEvent which is dispatched on any changes in buddy list. For example, if a person is removed from ST buddy list, the user manager also removes the person and the corresponding Facebook object from it's memory.

FacebookContactList maintains the logged in user's friends list as an array of user ids (in

```
String email = user.getField(ProfileField.PROXIED_EMAIL);
if (email == null) {
    email = user.getField(ProfileField.FIRST_NAME)
            + Messages.user_name_seperator //$NON-NLS-1$
                + user.getField(ProfileField.LAST_NAME)
                    + Messages.email_tail_string; //$NON-NLS-1$
}

person = PeopleUtil.getPerson(FacebookUtil.createContactId(userId),
        CommunityUtil.getCommunityMgr().getDefaultCommunity().getId());
String name = user.getField(ProfileField.FIRST_NAME) + " " //$NON-NLS-1$
        + user.getField(ProfileField.LAST_NAME);
person.setDisplayName(name);
DirectoryInfo dirInfo = person.getDirectoryInfo();
if (dirInfo == null || dirInfo.getSize() == 0) {

    dirInfo = new DirectoryInfoImpl();
    person.setDirectoryInfo(dirInfo);
    dirInfo.put(DirectoryInfo.NAME, name);
    dirInfo.put(DirectoryInfo.COMPANY, user
            .getField(ProfileField.WORK_HISTORY));
    dirInfo.put(DirectoryInfo.MAIL_ADDRESS, email);
    dirInfo.put(DirectoryInfo.IMAGE_PATH, user
            .getField(ProfileField.PIC));
}

if (refreshStatus) {
    String status = client.getOnlineStatus(user);
    ((                ) person).          (FacebookUtil
            .convertFbStatusToSTStatus(status));
}

((          ) person)
        .                t(user.getField(ProfileField.MESSAGE));
getUserMgr().putPerson(userId, person);
                          (person, true);
          .getPeopleService().addPersonListener(getUserMgr());
```

Figure 6.8: Code that creates ST person objects using user id

the format that Sametime recognises). It is responsible for getting friends of the logged in user and populates the user manager with corresponding ST person objects. It also resets the status of all Facebook contacts to 'Offline' when user logs out. As it knows the friends of the logged in user, Facebook manager's timer task that refreshes the status of friends uses this class to do so.

FacebookUser is a simple class that stores the set of user properties or attributes and serves them as required. It acts like a simple Java bean. Each user attribute is stored as a ProfileField class supplied by the Facebook API. Facebook gives out many attributes for a user which are all not supported by ST and at the same time, Facebook does not support or expose all attributes that Sametime supports. Attributes including company name, email address, user image and status are supported by both ST and Facebook. Name and telephone number are supported by Sametime but not Facebook. Attributes including first name, last name, age, address and work history are supported by Facebook but not ST. So last name and first name are concatenated to get name attribute. Email address is not exposed for the Facebook users but for some users, who granted email permission to the Facebook application, Facebook generates some proxied email addresses to use while sending email notifications. This is discussed in section 6.7. But just to show where email address

appears in business card, a mock email address is created for all users, except for those who have proxied email addresses, using the format *<firstname>.<lastname>@facebook.com*

FacebookClientWrapper is a proxy class for the Facebook client supplied in the Facebook library. It contains methods that are not provided but required by the application. It utilises the client functionality where sufficient and extends it as required. So it also acts as a decorator to the supplied client. It mainly parses and interprets the responses sent by Facebook platform. This class along with Facebook library acts as a bridge between sametime and Facebook. Facebook identifies users using their user ids also called UIDs (unique id) and stores them as Java 'long' type literals and user attributes as ProfileFields which is of type 'enumeration constant ' in Java. There is a constant in enumeration, ProfileField, which represents a single user attribute. So Facebook requires these (UID and/or ProfileFields) values to get any user related data. This class thus uses Facebook language to retrieve required information. For example, when FacebookContactList wants to load the friends of logged in user, then it sends the UID of logged user from user manager and the client wrapper uses the code in figure 6.9 to get the list of UIDs of friends of the logged in user.The method getFriendsAsList() parses the response and returns the list of UIDs. The client wrapper then requests for the user details as another batch request. The response format for each user is shown in figure 6.10. The actual batch response includes

```
try {
    beginBatch();
    friends_get();
    Document doc =(Document)((List) executeBatch(true)).get(0);
    if(doc!=null){
        doc.normalizeDocument();
        stripEmptyTextNodes(doc);
        friends = getFriendsAsList(doc, friends);
    }

    return friends;
} catch (FacebookException e) {
    // TODO Auto-generated catch block
    friends = null;
    MessageDialog.openError(Display.getDefault().getActiveShell(),
            Messages.generic_error_title,Messages.error_get_friends+
                e.getLocalizedMessage()); //$NON-NLS-1$ //$NON-NLS-2$
    e.printStackTrace();
} catch (IOException e) {
```

Figure 6.9: Code that gets the friends of logged in user

the user information for all the friends. So a recursive function is used to parse this batch response and thus create Facebook user objects (of type FacebookUser).

The client wrapper also gets the online status of multiple users in one go using a batch request. But this time, it uses FQL instead of REST API. The corresponding method takes in an array of ST user objects (of type Person) and uses a list of user ids that Facebook understands in a query to get the online status. Relevant code that does this function is shown in the figure 6.11 along with the response format of online status of a user. The figure also shows how the status is extracted from the xml response sent by Facebook and sets the status to the ST user objects. This class also provides other overloaded methods for getting user information.

```
<user>
        <uid>[redacted]</uid>
        <pic>http://profile.ak.facebook.com/v223/282/[redacted].jpg</pic>
        <proxied_email xsi:nil="true"/>
        <last_name>[redacted]</last_name>
        <work_history list="true">
        <work_info>
                <location>
                <city>[redacted]</city>
                <state>[redacted]</state>
                <country>[redacted]</country> </location>
                <company_name>[redacted]</company_name>
                <position>Software Engineer</position>
                <description>[redacted]</description>
                <start_date>[redacted]</start_date>
                <end_date/>
        </work_info>
        </work_history>
        <status>
                <message>had a tiring day.</message>
                <time>1228517699</time>
                <status_id>[redacted]</status_id>
        </status>
        <first_name>[redacted]</first_name>
</user>
```

Figure 6.10: Response format for user information

FacebookMenuManager extends SNViewActionMenuManager which is a menu manager used to populate the actions in status tool bar of ST connect client. This class provides the menu related to Facebook and is displayed as shown in the figure 6.12. In this implementation, it provides options to login, logout, add logged in user to the buddy list, get friends of the logged in user and edit Facebook status message. When users are not logged in, it disables all other options except login and when they log out, it enables all options except login. It communicates with FacebookManager to know if user is logged in and FacebookManager checks the session information and returns the login state of the user.

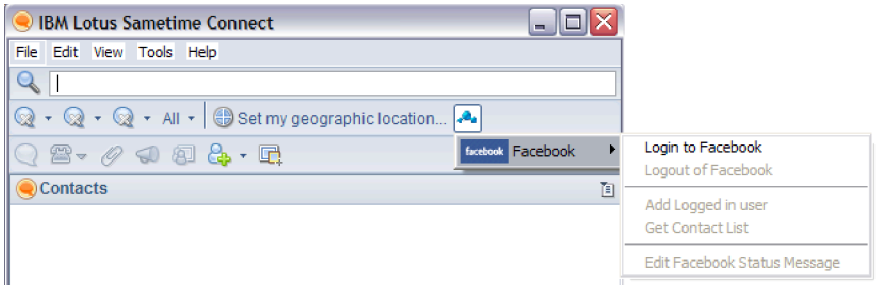


Figure 6.12: Facebook Menu in Status tool bar

FacebookGroupAction extends SNGroupAction to provide a menu for Facebook group actions. Facebook group actions are those actions that can be executed on a group of Facebook users. In this implementation, this menu provides options to refresh contacts, send notifications and send emails to the group of Facebook users. The group in this con-

text is a ST group which holds Facebook users saved as ST user objects. It uses a jface IMenuCreator to create a menu for all relevant actions. The group actions implemented in this module are

- Refresh Status of Facebook users
- Send Notification
- Send Notification Email

The generic and facebook group actions are shown in the figure 6.13.

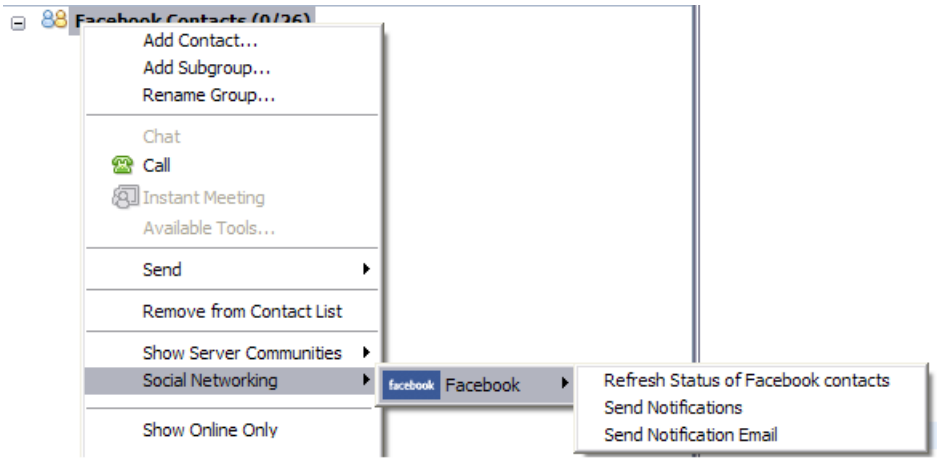


Figure 6.13: Cascading Group Actions

FacebookPersonAction extends SNPersonAction to provide a menu for Facebook person actions. Facebook person actions analogous to Facebook group actions are those actions that can be executed on a live name (a ST user shown in the buddy list. User names added in the buddy list are called live names because they show the status of the person visually by highlighting the name along with a status icon). The corresponding UI is seen as menu or a menu item in the context menu of a live name. In this implementation, this class provides a menu for Facebook whose menu items are the actions that can be executed for the Facebook user that the live name represents. The person actions implemented in this module are

- Facebook Chat
- Refresh status of the person
- Send Notification
- Send Notification Email

The generic and facebook person actions are shown in the figure 6.14.

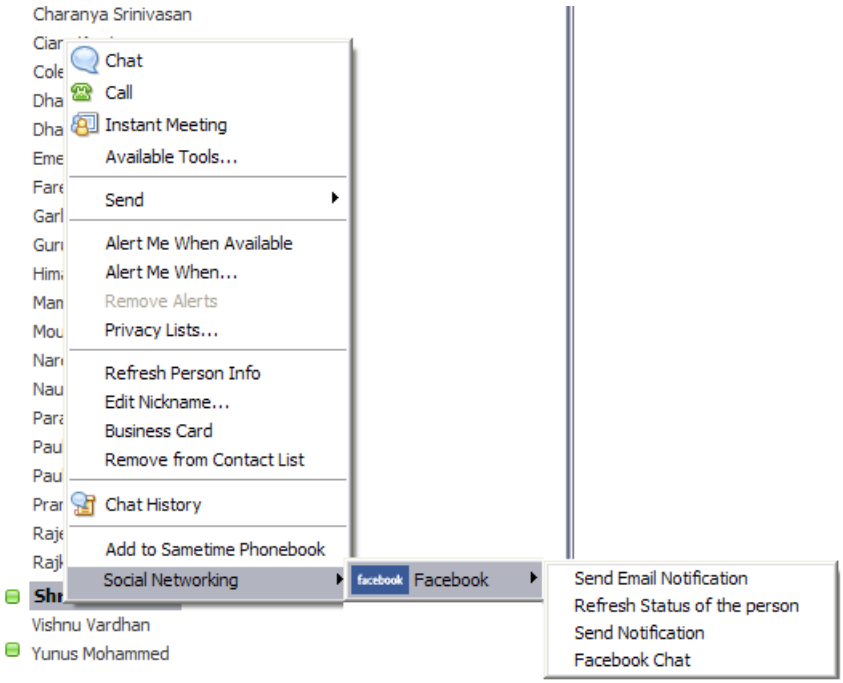


Figure 6.14: Cascading LiveName context menu

Facebook chat opens an SWT browser which links to the Facebook chat interface. The reason for not using ST chat interface is explained in the section 6.7. Figure 6.15 shows how this UI looks like.

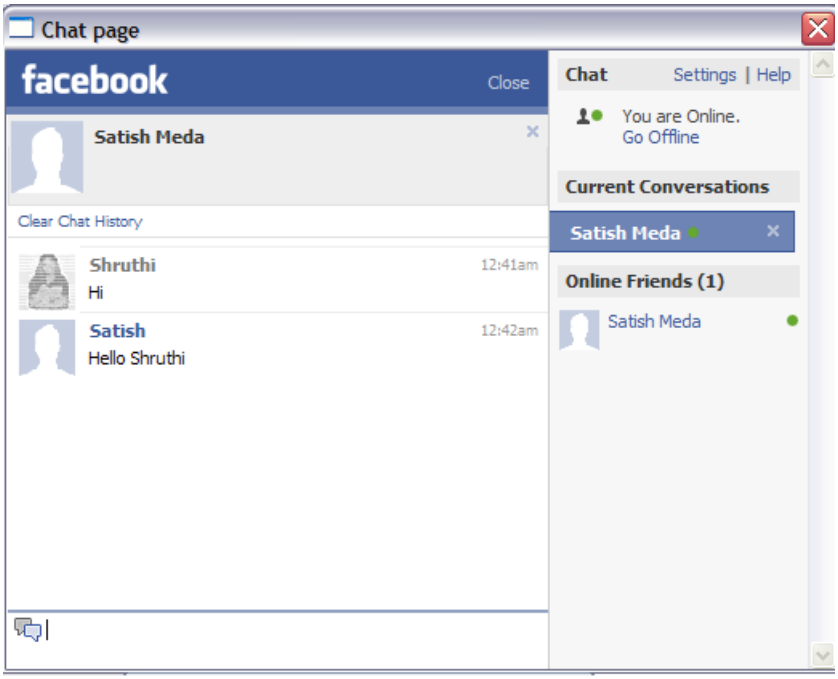


Figure 6.15: Facebook Chat

FacebookPreferences manages the user preferences for Facebook. This class extends class, FieldEditorPreferencePage and implements interface, IWorkbenchPreferencePage from Eclipse to provide a preferences page in ST preferences and manage them. The preferences used are

- A Boolean[1] preference which allows users to select a check box if they need integration of Facebook into ST.
- A String[2] preference which allows users to select a name for the group in buddy list to hold the Facebook contacts.
- A Integer[3] preference which allows users to set the time period for refreshing status of contacts. The necessity of this preference is explained in the section 6.7. This preference needs time in minutes and will be in effect only from the next user login.
- Another Integer preference which allows users to set the time period for session or connection time out. The necessity of this preference is explained in the section 6.7. This preference also needs time in minutes and will be in effect only from the next user login.

  Eclipse uses a preference store to store all the preferences set by the user (or default values if user did not change them), which are then retrieved as required in the application. The code that creates the preference fields is shown in the figure 6.16.

```
/**
 * Creates the field editors. Field editors are abstractions of
 * the common GUI blocks needed to manipulate various types
 * of preferences. Each field editor knows how to save and
 * restore itself.
 */
public void createFieldEditors() {

    addField(
        new BooleanFieldEditor(
            FacebookConstants.PREF_FACEBOOK_INTEGRATION_NEEDED,
            Messages.pref_fb_intg, //$NON-NLS-1$
            getFieldEditorParent()));

    addField(
            new StringFieldEditor(
                FacebookConstants.PREF_FACEBOOK_GROUPNAME,
                Messages.pref_fb_group_name, //$NON-NLS-1$
                getFieldEditorParent()));
    addField(
            new StringFieldEditor(
                FacebookConstants.PREF_CONNECTION_TIMEOUT,
                Messages.pref_fb_connection_timeout, //$NON-NLS-1$
                getFieldEditorParent()));
    addField(
            new StringFieldEditor(
                FacebookConstants.PREF_REFRESH_STATUS_PERIOD,
                Messages.pref_fb_refresh_status_period, //$NON-NLS-1$
                getFieldEditorParent()));
}
```

Figure 6.16: Facebook Preference Field Editors

Figure 6.17 shows how the preference page looks like.

[1]Boolean: A data type which takes either “true” or “false” as values
[2]String: A data type which takes any charanters as values
[3]Integer: A data type which takes numbers as values

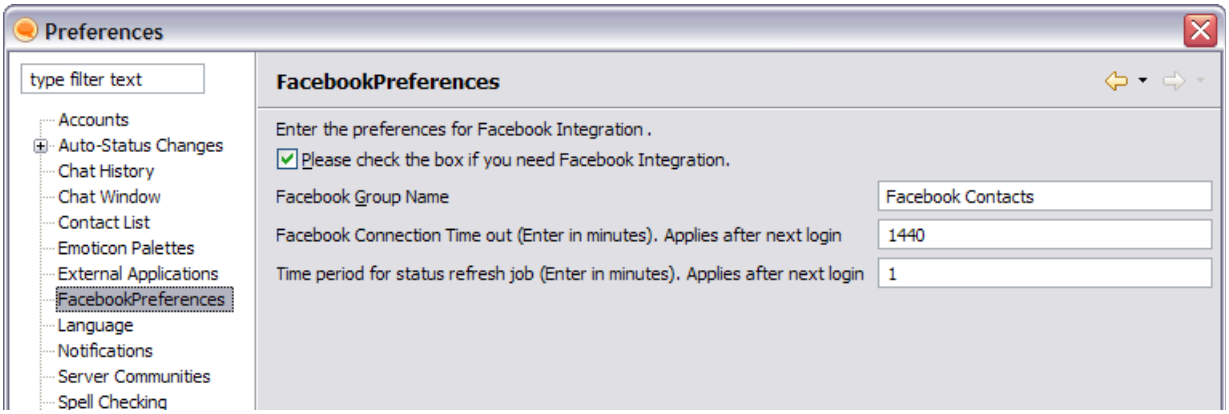


Figure 6.17: Facebook Preferences

6.6 Localisation

Localisation is done using message bundling mechanism of Eclipse framework. Eclipse provides a method to easily localise strings. It requires a class to extend org.eclipse.osgi.util.NLS class and the subclass should initialise the strings on class load. This subclass contains all the messages as class variables and these variables are given values in language specific properties files using the format *<key> = <value>*. The subclass of NLS is named as Messages.java for the project which contains all the strings used in Facebook implementation module and the messages.properties has the string literals. In real time project scenario, the translation team takes the messages.properties file and translates all the strings into different languages and puts them in different files which are named after the locale. For example, french strings will be placed in messages_fr.properties and english strings in messages_en.properties. Figure 6.18 shows the sample entries of messages.properties. Strings in plugin.xml are also localised and the corresponding entries are placed in files named as plugin_en.properties based on the locale and the strings are referred as *%<key>* where as entry in properties file looks like *<key> = <value>*.

Messages.java

```
public final class Messages extends NLS  {

    private static final String BUNDLE_NAME =
            "com.dit.st.sn.integration.Facebook.Strings.messages";//$NON-NLS-1$
    //Group Action Strings
    public static String action_refresh_status;
    public static String send_notif_email;
    public static String send_notif;

    //View Action Strings
    public static String add_me;
    public static String edit_my_status_msg;
    public static String get_friends;
    public static String logout;
    public static String login;

  static {
      initializeMessages(BUNDLE_NAME, Messages.class);
  }
```

messages.properties

```
# Group Actions
action_refresh_status=Refresh Status of Facebook contacts
send_notif=Send Notifications
send_notif_email=Send Notification Email

#View Actions
add_me=Add Logged in user
edit_my_status_msg=Edit Facebook Status Message
get_friends=Get Contact List
logout=Logout of Facebook
login=Login to Facebook
```

Figure 6.18: Sample entries in Messages.java and corresponding values in messages.properties

6.7 Some Challenges faced

This section covers discussion around some issues faced and how they were resolved in development phase. The issues were related to the incompatibilities between the requirements and relevant support provided from Facebook library and API. Some of the issues were also resolved by changing the open source library used. So the source package is used but not just the jar files.

- **Posting requests using the client library**

  For any Facebook request, the REST client from Facebook client library sends HTTP (**H**yper **T**ext **T**ransfer **P**rotocol) Post requests through a java.net.HttpURLConnection to Facebook. It then parses the document to log the response, returns the same document to the application, then the application parses and interprets the response.

  - **Problem:** But it uses org.apache.commons.io.input.AutoCloseInputStream which closes automatically once the end of file is reached. So the client throws a premature end of file exception whose stack trace is shown here. So no responses are parsed.

```
Premature end of file.
org.xml.sax.SAXParseException: Premature end of file.
at org.apache.xerces.parsers.DOMParser.parse(Unknown Source)
at org.apache.xerces.jaxp.DocumentBuilderImpl.parse(Unknown Source)
at javax.xml.parsers.DocumentBuilder.parse(Unknown Source)
at
com.facebook.api.FacebookRestClient.callMethod(FacebookRestClient.java:
266)
at
com.facebook.api.FacebookRestClient.callMethod(FacebookRestClient.java:
224)
at
com.facebook.api.FacebookRestClient.auth_createToken(FacebookRestClient
.java:1069)
```

- **Solution:** The REST client is changed to use org.apache.commons.httpclient.HttpClient and org.apache.commons.httpclient.methods.PostMethod. Once the method is executed, response is read as xml instead of parsing the stream directly. The old and new code is shown in the figure 6.19 as shown in a difference merge tool.

- **Recognising ST person objects as Facebook users**

  When ST person objects are added to the ST buddy list, there should be someway of recognising a user as Facebook user in order to display Facebook relevant person actions in the context menu of a livename. A livename is a term used to refer ST person in buddy list because it is designed to style the appearance of name and the status icon based on the online status of the corresponding user. One way of achieving this is to set some data in person object that can be retrieved for analysis at later stages. Data like type of the user or SNname that the user belongs to serves the purpose. Data can be set using *person.setData("key","value")* method call. So by using *person.setData("type", "Facebook")* and again *person.getData("type")*, the application should be able to recognise the person object involved as a Facebook user.

  - **Problem:** Once the client is restarted, the data stored using *person.setData()* is lost. On investigation, it is found that ST does not save the data set on person objects into the local cache nor does it save remotely. It is only stored in the memory. So *person.getData("type")* does not suffice the purpose. So the application should explicitly store it locally and use it at later stages. But as storing locally means, making it mandatory for users to use a single computer always. So this is not analogous to ST which allows user to login from any computer.
  - **Solution:** ST uses contact id along with community id as an attribute to uniquely identify a person object. Making use of this design, the application now creates it's own contact id and provides to ST, where the created contact id is made up of a static string to recognise as Facebook user and the UID of the corresponding Facebook user which is provided by Facebook platform itself. For example, if a Facebook user has a UID of 12345678, then the contact id used while creating ST person object will be "SNFB-UID12345678". When resolving Facebook users from ST person objects, the contact id is parsed for the static Facebook identifier string "SNFB-UID", and if found, the rest of the contact id string is used as UID to get the Facebook user from user manager. The UID field is not explicitly given as an enum constant in enum type, ProfileField, and so is added to it.

- **Chat with Facebook users (stored as ST users) as with actual ST users**

  Chat is the feature around which Instant Messaging systems are built. This feature of an IM system, allows two online users to send/receive messages to each other almost in realtime. The requirements gathered from the survey mentioned in section 4.2.1 include the chat feature as most wanted feature for the users surveyed. The ideal scenario would be to double click on the live names which opens a chat interface as with actual ST users. But the problem with this approach is that, chat interface opens up a connection with the community server for sending and receiving chat messages, so it requires the users at both ends of a chat to be members of the community which is not the case. Hence it is not possible to provide that level of integration. A transcript of chat with an IBM expert in this area of ST is provided in Appendix-A. Another alternative is to use the Facebook - LiveMessage API. This API needs some javascripts to register an event for chat functionality

  - **Problem:** The Facebook API requires a JSON object of the form { "message":"abcd" }. There was a bug in the client library which returns an error code - 0 but error message - success. This error code was not defined in the error code list in the Facebook developers wiki (Facebook Developers Wiki 2008*a*). Hence a bug was reported at their website (Google Code 2008, Google Code: Issues 2008).
  - **Solution:** As the API can not be used until it is fixed, a workaround was implemented which suffices the purpose. The workaround is using the Facebook's Chat interface embedded in an Eclipse browser (which gives an impression that the window is part of ST platform). But a drawback of this workaround is that when the Facebook chat interface is opened, it shows all online friends and allows chatting with them where as the context of using this feature is to chat with the person with whom the action(chat action) is invoked. If this feature is moved to the group's context menu instead of live name context menu, then from user point of view, that would mean that the action allows user to start a multi person chat with the group members which is not the intention.

- **Sending Emails**

  As part of IM system, the application should allow users to send emails to their friends. For sending an email, the application should know email addresses of the recipients. Unlike other attributes of a user, email address is not provided by Facebook. Eventhough, there is a place holder in ST Person object, Facebook does not send email addresses of it's users to other users because of it's privacy policy. So it uses proxied email addresses which are generated for application/user pair. But it needs users at both ends to grant permissions to the Facebook application.

  - **Problem:** Facebook API allows sending emails to users via the Facebook application. So users at both ends must grant the application required permission(an extended permission called "email"). So when user A tries to send an email to user B, the request first goes to the application, then it checks if user A has granted it email permission, if yes then it checks if user B has granted it the "email permission", if yes then it sends the email. However if user A and/or user B have not granted the "email permission", then it throws an error to user A. If user A has granted the permission but not user B then it sends a grant permission request to user B. But in case of

desktop applications, there is no way to send a grant permission request to other users.

- **Solution:** Not complete but partial solution has been implemented. When user A tries to send an email but did not grant email-permission to the Facebook application then Facebook throws a permission error. Now the application presents a grant permissions dialog to user A and when it is granted, it tries to send the email to user B. If user B has not granted the "email permission", then user A will be informed about that. Now user A has granted "email permission" to the Facebook application. When the same situation happens to user B as well, then they will be able to send emails to each other. Unfortunately, this is not the real solution, this is the only possible way for desktop application unless Facebook provides new API to send grant permission requests to other users. An enum constant named PROXIED_EMAIL in enum type, ProfileField, whose value is "proxied_email", is added in the Facebook client library.

- **Status of a Facebook user**

  There are two concepts of status of a user in Facebook. One is the online status and the other is the status message set by the user. The online status is the availability status of the user which is similar to an IM system and the status message is the message set by the user to let the profile visitors know what the user is doing. Besides using the status message field for setting the actual status message, SN users use it to set their favourite captions or any announcements to all the friends or wishing the friends on a common event. But ST has only one status which is availability or online status. So mapping between Facebook status and ST status is not straightforward as setting other user attributes like user name, company name. The application is responsible for matching between ST and Facebook status. It uses a Java enum type to hold the Facebook online statuses ("active", "offline", "idle" , "error") and then uses an utility method that converts Facebook status to ST status using the mapping shown in the table 6.2. Although ST provides many more statuses, Facebook provides only those shown in the table 6.2.

Table 6.2: Online status mapping between ST and Facebook

Facebook status	**ST status**
active	STATUS_AVAILABLE
offline	STATUS_OFFLINE
idle	STATUS_AWAY
error	STATUS_UNKNOWN

ST also provides a way to set some status message to the ST person objects. Although in ST, there are predefined strings to set based on the online status, it also provides a way to set a user defined status message. Using *person.setStatusText()* sets the status message and can be seen on the business card of the user that the object *person* represents.

- **Problem:** The Facebook client library used does not provide an enum constant, in ProfileField enum type, for status message.
- **Solution:** A new enum constant is added in the enum type, ProfileField(available in the client library source package), named *Message* whose value is "message". This is a child attribute of the status tag in the xml sent by Facebook as shown in the figure 6.10.

- **When to Refresh Status of the contacts?**

  In an IM system, IM client changes the status of a live name when the status of the corresponding user changes. This is achieved using the event handling mechanism. Server sends the status change events to the client and the client updates the status.

  - **Problem:** But this application being a pure desktop one and is only connected to the Facebook server by REST API, no callbacks can be used to get any events. So the problem is how to refresh the status of Facebook users.
  - **Solution:** Status of the Facebook users, added as ST users, in the connect client can be refreshed periodically and users are allowed to set the period in preferences. The lower the period, the accurate the status. But performance is a considerable factor. So a default value of 3 minutes is used and users are allowed to refresh the status when ever they need by providing a group action. Although this is not an ideal solution, it is a reasonable solution considering the fact that users would not constantly monitor the status and if they want accurate status, they can always refresh the status manually. The application runs a daemon thread after login action, to refresh the status of the logged in user's friends.

- **Session timeout**

  This issue is similar to the issue of refreshing status of contacts. Generally, in a web application, when user session times out, requests made after timeout are not processed at the server and a time out error is thrown. Even in Sametime Social, requests made after session timeout, are not processed and the user is thrown an error. To avoid sending requests after session times out, a daemon thread is run which simulates the time out operation and once the session times out, user is warned about it and is asked to login. This avoids sending and receiving requests which are not at all processed at the server. Users are also allowed to set the session timeout period as a Facebook preference. This session timeout period is sent to Facebook while creating a client, and is also saved locally for use by the daemon thread. But the parameter sent at creation time did not seem to have an effect so the application sets this value explicitly after the session is created.

6.8 Conclusion

The design models created in chapter 5 were used in this chapter to create the software. A working and fully functional plugin was created in this chapter. All functionalities, represented as use cases in chapter 4, were provided by this plugin when installed in a Sametime Connect Client. This plugin, in this first version, has provided integration with Facebook but has also provided the framework which could be used for integration of more SNs. Some important decisions made in development phase, significant code snippets from the plugin and some challenges faced while implementation were also covered in this chapter. Having developed the software, the next phase would be to test it, which is covered in chapter7.

```
for (int i = 0; i < people.length; i++) {
    ids[i] = new Long(FacebookUtil
            .extractUidFromContactId(((PersonImpl) people[i])
                    .getContactId()));
}
try {
    MyLongArray uidArray = new MyLongArray(ids);
    String uidsArray = uidArray.toString();
    beginBatch();

    fql_query("SELECT uid,online_presence FROM user WHERE uid in "
            + uidsArray + " AND online_presence"); //$NON-NLS-1$ //$NON-NLS-2$
    List batchResponse = executeBatch(true);
    Document doc = (Document) batchResponse.get(0);

    doc.normalizeDocument();
    stripEmptyTextNodes(doc);
    NodeList nl = doc.getElementsByTagName(TAG_USER); //$NON-NLS-1$
    int length = nl.getLength();
    List<Long> uidList = new ArrayList<Long>(Arrays.asList(ids));
    for (int i = 0; i < length; i++) {
        extractStatusForUser(nl.item(i), uidList);
```

Response format for online status of single user

```
<user>
  <uid>1286358545</uid>
  <online_presence>offline</online_presence>
  <status>
   <message>is just passing time.</message>
   <time>1228519484</time>
   <status_id>38538782077</status_id>
  </status>
 </user>
```

Code that extracts status from facebook response

```
if (doc.getNodeName().equalsIgnoreCase(TAG_USER)) { //$NON-NLS-1$
    NodeList children = (doc.getChildNodes());
    for (int i = 0; i < children.getLength(); i++) {
        if (children.item(i).getNodeName().equalsIgnoreCase(
                ProfileField.UID.fieldName())) {
            uid = children.item(i).getTextContent();
            if (pids.contains(new Long(uid))) {
                Node n = children.item(++i);
                if (n.getNodeName().equalsIgnoreCase(
                        TAG_ONLINE_PRESENCE)) { //$NON-NLS-1$
                    status = n.getTextContent();
                }
                n = children.item(++i);// Goes to <status>
                if (n.getNodeName().equalsIgnoreCase(TAG_STATUS)) {
                    NodeList statusTag = n.getChildNodes();
                    for (int j = 0; j < statusTag.getLength(); j++) {
                        if (statusTag.item(j).getNodeName()
                                .equalsIgnoreCase(
                                        ProfileField.MESSAGE
                                                .fieldName())) { //$NON-NLS-1
                            status_msg = statusTag.item(j)
                                    .getTextContent();
                        }
                    }
                }
                Person person = fbMgr.getUserMgr().getPerson(uid);
                if (person != null) {
                    ((          ) person).          (FacebookUtil
                            .convertFbStatusToSTStatus(status));
                    ((          ) person).          (status_msg);
                }
```

Figure 6.11: Code that gets the online status of a list of users

```
HttpClient postClient = new HttpClient();
PostMethod post = new PostMethod(serverUrl.toString());
for (Map.Entry entry : params.entrySet()) {
        String key = entry.getKey().toString();
        String value = entry.getValue().toString();
        post.addParameter(key, value);


}
String rawXml = "";
try {
        int responseCode = postClient.executeMethod(post);
        if (responseCode >= 200 && responseCode < 300) {
                // TODO: Replace with streamed reads
                rawXml = post.getResponseBodyAsString();
                System.out.println(rawXml);
                // parseResponse(rawXml);
        }
} catch (HttpException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
} finally {
        post.releaseConnection();
}


return rawXml;
```

New Code

```
    HttpURLConnection conn = (HttpURLConnection) serverUrl.openConnection();
    if ( this._timeout != -1 ) {
            conn.setConnectTimeout( this._timeout );


    }
    if ( this._readTimeout != -1 ) {
            conn.setReadTimeout( this._readTimeout );
    }

    try {
            conn.setRequestMethod( "POST" );


    }
    catch ( ProtocolException ex ) {
            System.err.println( "huh?" );

            ex.printStackTrace();


    }
    conn.setDoOutput( true );
    conn.connect();
    conn.getOutputStream().write( buffer.toString().getBytes() );

    return conn.getInputStream();
```

Old Code

Figure 6.19: Changed Post requests

Chapter 7

Testing

7.1 Introduction

The fully functional software was resulted at the end of implementation phase, which is described in the chapter 6. This software has to be tested to make it error free. Testing for the project is based on the guidelines provided by the IEEE standard 829 (*IEEE Standard for Software and System Test Documentation* 2008). Testing is an important phase in software development life cycle because, software is generally prone to a lot of bugs which would have not been anticipated at the time of development. So an important framework is required to perform testing of a product. The standard provides a common framework for all the activities, tasks and processes that support the software life cycle. It recommends to define the test tasks, and the required inputs and outputs to run the test. It also recommends to define the master test plan and level test plans along with the related test documentation to include the test design, test case, test procedure and test reports.

7.2 Testing

Although the guidelines provided by IEEE 829 are very formal and suit for real time corporate projects, the essence of the standard can be adapted to small sized projects. The section 7.2.1 presents the Master test plan of the project.

7.2.1 Test Plan

The main objective of the master test plan is to do the system testing for the project and deliver the functional system free of bugs. Test cases are planned to execute in each logical module from the user's perspective. For example in an application that integrates ST and SN through a common API, the logical modules from user's perspective can be authentication module, chat module, email module, common UI irrespective of number of SNs used and preferences module. The test cases planned would cover all these modules. Section 7.2.2 presents the test cases executed for the project based on IEEE 829 test case format. The application delivered only supports Facebook in this version, so test cases are also based on ST and Facebook.

7.2.2 Test cases

Here are some of the test cases executed to test the application functionality and usability. The test cases are categorised as follows based on the context of the application where a particular feature applies. Each test case is discussed and executed against the application developed and is documented in this section. All test cases have the following common prerequisites and the special prerequisites that apply only for the test cases are mentioned in test case descriptions.

1. Sametime connect client should be launched.
2. Facebook application is created at Facebook platform.
3. This software is installed and configured properly.

7.2.2.1 Authentication Test Cases

Test cases related to authentication module of the application are discussed and executed in this section. The scenarios related to login and logout are thus discussed here as follows.

1. **User cant logout before logging in**

   Description: A Facebook user should be allowed to login and then logout but not the other way.

   Prerequisites: None

   Steps to execute:

   (a) Click on "Social Networks Actions" on status toolbar.

   (b) Select the appropriate social network menu. For example select 'Facebook' from the pull down menu.

   Expected Result:

   (a) As user is not logged in, "Login" menu option should be enabled and others disabled.

   (b) Clicking on the Login menu option, the user should be able to login to the Facebook using valid user credentials.

   (c) After user successfully logs in, login menu item should be disabled and others enabled.

   Actual Result:

   (a) Login option enabled initially.

   (b) Clicking on the Login menu item, user is presented with login dialog. When correct user credentials are entered, user is informed about the success of login and allowed to return to the application.

   (c) After successfully logged in and closed the login dialog, the login menu changes disabling login menu item and enabling other menu items which are valid only after user is logged in.

   Result: Pass

2. **User relogin after login, re-logout after logout**

   Description: Once User is logged in, there is no need to login again. But in case the user logs in again, the user should be informed about the action. Similarly user logout

action does not need to be repeated and the user should be informed about any repetitive logouts.

Prerequisites: None

Steps to follow:

(a) The ST user who also have Facebook account logs in successfully into Facebook using application's login UI.

(b) The same user tries to login to Facebook again.

(c) User logs out successfully

(d) The same user tries to logout again.

Expected Result:

(a) User should not be allowed to login for the second time or when tries to login, should be informed about a live session.

(b) Similarly when users try to logout for the second time, either they should not be allowed to logout or be informed about the invalid action.

Actual Result:

(a) User logs in successfully using the application's login UI

(b) Same user tries to login for the second time, but login UI is disabled.

(c) User logs out successfully

(d) When user tries to logout again, logout menu item is disabled.

Result: Pass

3. **Login using invalid credentials**

Description: Invalid credentials should not authorise Facebook application and so user should not any get Facebook services.

Prerequisites: None

Steps to execute:

(a) Click on the SN Actions>Facebook>Login menu item in status tool bar.

(b) Enter wrong credentials

(c) Try to refresh the status of the Facebook group members or any action that needs Facebook user session (which needs user to be logged in successfully)

Expected Result:

User should not be allowed to perform any actions that need a session or inform user of absent session or allow user to login again.

Actual Result:

Result: Pass

7.2.2.2 Status Test Cases

1. **Status change of Facebook users reflects in the contact list**

   Description: The status changes of Facebook users in the website should be reflected in the buddy list of Sametime Connect client.

   Prerequisite: User should be logged into both Sametime and Facebook.

   Steps to Execute: The steps to execute this test case is outlined in the table 7.1. The table shows the verification points for the test case along with the expected and actual result. The status to check includes the online presence as well as the status message. Online presence is checked visually by seeing the status icon in livename and the status message is checked in the business card.

Table 7.1: Test case 4-Status change reflection

Verification Points	Expected Result	Actual Result
Real time Status Reflection	• Change the status of a user in Facebook website. • The user status should be changed in the connect client almost at realtime	The status is not changed immediately. But this is explained in the section 6.7
Refresh Status from the UI	The UI for refreshing the status of the contacts is designed at two locations of the connect client. • In context menu of a Facebook group • In context menu of a Facebook livename as part of refreshing person information	• User selects Refresh Status menu option from the context menu of a Facebook group and it refreshes the status of all members of the group successfully. • User selects Refresh status menu option from context menu of a Facebook live name and it refreshes the status of the selected person successfully.
Refresh Status Daemon thread	• Status of the contacts is refreshed periodically and the period is set as a preference.	• Status is refreshed automatically and periodically where the period is taken from preferences.

2. **Change the Status message of logged in user**

   Description: Users should be able to change their status message from the application's UI.

   Prerequisites: User should be logged in both ST and Facebook though the application's UI.

   Steps to execute:

   - Click on the *Edit my status message* menu item in status tool bar.

- Enter the new status message in the Edit Status dialog.

Expected Result:

- Status message is changed in the Facebook website.
- The new status message is also shown in the business card.

Actual Result:

New status message is reflected in the Facebook website as well as in the buddy list.

Result: Pass

7.2.2.3 Preference Test cases

1. **Session times out using the timeout value in preference page**

   Description: As already explained in section 6.7, session timeout preference value is sent to the server. So session should time out after this amount of time. This scenario is checked in this test case.

   Prerequisites: User should be logged in.

   Steps to execute:

   - User enters the preference related to session timeout in Facebook preferences page.
   - Observe if session times out after the amount of time saved in preferences.

   Expected Result:

   - Session times out after the amount of time saved in preferences.
   - All Facebook menus should be disabled except *Login* menu item.

   Actual Result:

   - A dialog pops up after a period of time, specified by session time out preference, warning user about the session timeout.
   - After session times out, all Facebook menus are disabled. They are enabled only after user logs in again.

   Result: Pass

7.2.2.4 Test cases about integration

1. **Populating friends list and user fields properly**

   Description: The main point of integrating SN and ST is to have one interface for both systems. So it is required to add all the friends in SN to the ST. The added list should also have as much information related to the users as possible. The supported attributes as mentioned in section 6.5 are name, company name, email address for those users with proxied email addresses and user image.

   Prerequisites: User should be logged in.

   Steps to execute:

   - Select the *Get Contact List* menu item from the *Facebook* menu in the status tool bar.

- Check the buddy list.

Expected Result:

- All the friends of the logged in user should have been added to the buddylist in a group named by the *Facebook Group name* preference value.
- When hovered over each live name in this group, the business card should contain correct attributes.

Actual Result:

- Buddy list contains all the friends of logged in user.
- Business card contains all attributes of user mentioned in the description of this test case.

Result: Pass

2. **Send Emails to the friends**

   Description: Users should be able to send emails to their friends through the ST interface.

   Prerequisites: User should be logged in and have atleast one friend added to the buddy list.

Verification Points	Steps to execute	Expected Result	Actual Result
Send Group email	• Select the *Send Notification email* menu item from the context menu of the Facebook group. Or • Another way of sending emails is from *Send* menu in group's context menu. • Type the email content and send	• If senders have not granted access to Facebook application, it should inform them about it and should allow them to grant access. • If receivers have not granted permission, then the application should inform the sender about it. • If both sender and receiver have granted permission, then receiver should get the email sent.	Same as expected result.
Send personal email	• Select the *Send Notification email* menu item from the live name's context menu. Or • Another way of sending emails is from *Send* menu in live name's context menu. • Type the email content and send	• If senders have not granted access to Facebook application, it should inform them about it and should allow them to grant access. • If receivers have not granted permission, then the application should inform the sender about it. • If both sender and receiver have granted permission, then receiver should get the email sent.	Same as expected result.

Result: Pass

3. **Send Notifications to the friends**

   Description: Users should be able to send notifications to their friends through the ST interface.

   Prerequisites: User should be logged in and have atleast one friend added to the buddy list.

Verification Points	Steps to execute	Expected Result	Actual Result
Send Group notification	• Select the *Send Notification* menu item from the context menu of the Facebook group. • Type the notification content and send	• If senders have not granted access to Facebook application, it should inform them about it and should allow them to grant access. • If receivers have not granted permission, then the application should inform the sender about it. • If both sender and receiver have granted permission, then receiver should get the notification sent.	Same as expected.
Send personal notification	• Select the *Send Notification* menu item from the live name's context menu. • Type the notification content and send	• If senders have not granted access to Facebook application, it should inform them about it and should allow them to grant access. • If receivers have not granted permission, then the application should inform the sender about it. • If both sender and receiver have granted permission, then receiver should get the notification sent.	Same as expected

Result: Pass

4. **Chatting with friends**

   Description: Users should be able to chat with their online friends

   Prerequisites: User should be logged in. User should have atleast one online friend.

   Steps to execute:

- Select the *Facebook Chat* menu item from the live name's context menu.
- Start chatting.

Expected Result:

- Able to exchange instant messages with the friend selected.

Actual Result:

- See all online friends.
- When one friend is selected, able to exchange instant messages.

Result: Pass . Though not completely as expected. This is explained in section 6.7.

7.3 Conclusion

The software created in chapter 6 is tested using the guidelines provided by IEEE standard 829. The guidelines are briefly discussed in the section 7.1. A test plan is created and is implemented as described in section 7.2.1. The test cases were executed as explained in the section 7.2.2 and ensured that the software is error free and satisfy the requirements of the project.

Now that the software is created and tested, it has to be evaluated to know how well it works and this is covered in the chapter 8.

Chapter 8

Evaluation

8.1 Introduction

In any undertaking, evaluation would be the first and important step, so is the case with software. Evaluation of software quality determines the position of the software in the market and hence an important step in the software life cycle. Delivering a quality software is fundamental for it's success. But defining quality is a subjective matter. It is not necessary that all stakeholders agree on what they think is good quality. ISO (International Standards Organisation) (ISO 1986)defines quality as

> "*Quality is the totality of features and characteristics of a product or service that bear on it's ability to satisfy specified or implied needs*"

IEEE defines quality as

> "*Quality is the degree to which a system, component, or process meets specified requirements and customer or user needs or expectations"*

Both the definitions focus on customer satisfaction with the software product.

8.2 ISO/IEC 9126

ISO/IEC 9126 is a standard developed by ISO and the International Electrical technical Commission (IEC) for Software Engineering – Product Quality. Punter and others (Punter, Van Solingen & Trienekens 1997) presented the intensions of the ISO/IEC 9126 standard as

- To provide a specification and evaluation model for the quality of software products.
- To address user needs of a software product explicitly, by allowing a common language for user requirement specification, which is easily understandable by it's stake holders
- To evaluate software product quality based on observation but not opinion
- To make a quality evaluation reproducible.

The standard is published in four parts. First part (ISO/IEC 2001) identifies the software quality through six characteristics which are further divided in to sub-characteristics. Part 2 (ISO/IEC 2002*a*)describes the external metrics to measure the characteristics and sub-characteristics identified in part 1; part 3 (ISO/IEC 2002*b*) the internal metrics and part 4 (ISO/IEC 2002*c*) defines

the quality-in-use attributes. The standard (ISO/IEC 2001) states that the quality of the product can be evaluated by measuring the internal attributes or external attributes (measuring external behaviour) which are dependant on internal attributes or quality-in-use attributes which are again dependant on external behaviour. The characteristics and sub-characteristics identified by the standard are summarised in the table 8.1.

Table 8.1: Characteristics and their explanation from ISO 9126(source: (Chua & Dyson 2004))

Characteristic	Sub-characteristic	Explanation
Functionality	Suitability	Can software perform the tasks required?
	Accurateness	Is the result as expected?
	Interoperability	Can the system interact with another system?
	Security	Does the software prevent unauthorised access?
Reliability	Maturity	Have most of the faults in the software been eliminated over time?
	Fault tolerance	Is the software capable of handling errors?
	Recoverability	Can the software resume working and restore lost data after failure?
Usability	Understandability	Does the user comprehend how to use the system easily?
	Learnability	Can the user learn to use the system easily?
	Operability	Can the user use the system without much effort?
	Attractiveness	Does the interface look good?
Efficiency	Time Behaviour	How quickly does the system respond?
	Resource Utilisation	Does the system utilise resources efficiently?
Maintainability	Analysability	Can faults be easily diagnosed?
	Changeability	Can the software be easily modified?
	Stability	Can the software continue functioning if changes are made?
	Testability	Can the software be tested easily?
Portability	Adaptability	Can the software be moved to other environments?
	Installability	Can the software be installed easily?
	Conformance	Does the software comply with portability standards?
	Replaceability	Can the software easily replace other software?
All characteristics	Compliance	Does the software comply with laws or regulations?

8.3 Evaluation of Sametime Social

The evaluation for this product is done based on the format used by Chua and Dyson (Chua & Dyson 2004).

8.3.1 Evaluation Methodology

After completion, the project was given for use to a subset of people selected for the survey in requirements phase. The sample users chosen for evaluation were both ST and Facebook users. These users were categorised based on the attributes and behaviour (discussed in section 2.2.3.2) and charted using a bar diagram as shown in figure 8.1. From the bar diagram, it is evident that of the sample users, there were no Alpha Socialisers, Attention Seekers and Functionals. So feedback was collected from the Faithfuls and Followers and used for evaluating the project. Faithfuls in this context, as defined in section 2.2.3.2, would use this software to maintain their old relationships and thus wanted features that are relevant to that purpose (features like maintaining friends list, chatting , telephoning and sharing files with them). Followers on the other hand would use it for the sake of keep up with others and the technologies, thus wanted both basic and advanced features (including posting comments on walls, access other Facebook applications and control Facebook settings).

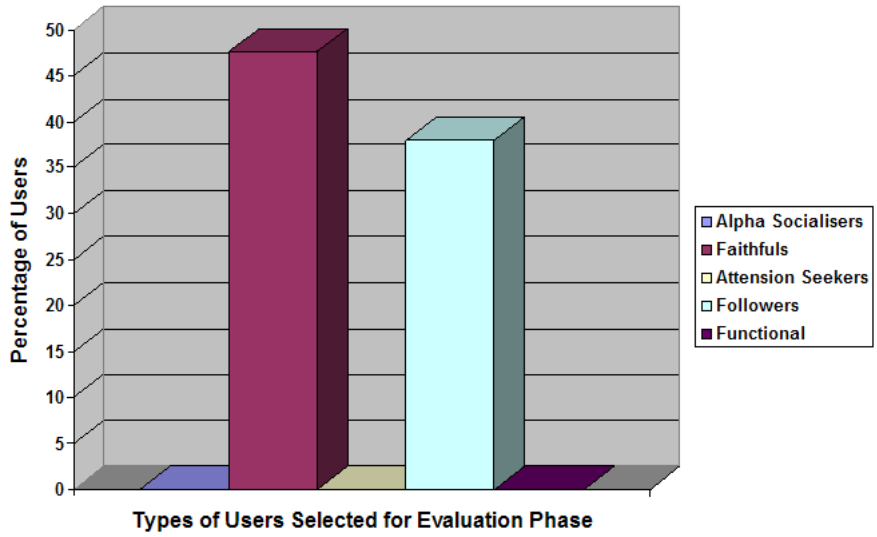


Figure 8.1: Bar Diagram to show types of users selected for feedback

The quality characteristics and sub-characteristics identified by the ISO 9126 standard form the basis for evaluation. But from naive user point of view, all characteristics are not easily assessable but "Functionality", "Reliability", "Usability", sub-charateristic of "Efficiency" - "Time Behaviour" , "Maintainability" except for its sub-characteristic - "Analysability", "Adaptibility" and Installability" sub-characteristics of "Portability" are assessed.

8.3.2 Results

The results are summarised in a matrix shown in table 8.2. Columns and sub-columns of the matrix are the characteristics and corresponding sub-characterstics. Rows are the key features of the product. An asterisk in the table cell means that the feature of that row satisfies the requirements of the sub-characteristic of that column. Numbers indicate the points in the explanation given in this section for not satisfying the requirements.

1. The features do not perform ideally. Section 6.7 explains how these issues are handled in the project.The term "User Integration" is used to indicate how well the Facebook users are integrated into ST. As explained in the section 6.5, all attributes of Facebook user do not have a match in Sametime. So all possible attributes are considered for integration. The term "Status Integration" is used to indicate how well the status of users in Facebook is mapped with that of corresponding ST users (corresponding ST users mean the ST user objects created and added to represent a Facebook user). The mapping is successful but the status is not refreshed in real time as expected. Chatting with Facebook users, who are added to ST, is not same as chatting with actual ST users which is also explained in section 6.7.

2. Chat feature for Facebook users, added as ST users, in this product allows the users to chat with any of their online friends where as they should be allowed to chat with the user for which the action is invoked.

3. ST also allows the users to change the preferences when offline. So authenticity can not be guaranteed.

Table 8.2: Evaluation of Sametime Social using ISO 9126

Features	Functionality				Reliability			Usability				Efficiency	Maintainability			Portability	
	Suitability	Accurateness	Interoperability	Security	Maturity	Fault Tolerance	Recoverability	Understandability	Learnability	Operability	Attractiveness	TimeBehaviour	Changeability	Stability	Testability	Installability	Adaptability
Authentication	*	*	*	*	*	*	*	13	*	*	8	10	*	11	*	*	*
Preferences	*	*	*	3	*	*	*	*	*	*	*	*	*	11	*	*	*
User Integration	1	*	*	*	*	*	*	*	*	*	*	10	*	11	*	*	*
Status Integration	1	*	*	*	*	*	*	*	*	*	*	10	*	11	12	*	*
Chat	1	2	*	*	4	*	*	6	*	*	*	*	*	11	12	*	*
Email	7	7	*	*	*	1,5	*	*	*	*	9	*	*	11	*	*	*
Notifications	*	*	*	*	*	*	*	*	*	*	9	*	*	11	*	*	*

4. Any changes or bug fixes in chat feature are not considered for incorporation in the project because of time constraints. The section 6.7 discusses the problem regarding chat feature.

5. Sending emails is allowed in Facebook applications, only when the sender and the recipient have granted email permission to the Facebook application.

6. Both the chat UI and the method to launch it are different for actual ST users and Facebook users added as ST users. This creates confusion for the user.

7. Emailing contacts is not implemented as expected due to the non availability of emails of Facebook users. This is discussed in section 6.7.

8. Facebook needs it's own login form to be used for log in. So it does not match the ST UI.

9. The UI used for Notifications, email notifications are not styled properly but a basic dialog is used due to time constraints. So not very attractive.

10. The time of execution for these features depend on the network speed and the number of users.

11. If Facebook changes the API, the Facebook implementation module might need some changes which might affect the stability of the product but not necessarily.

12. For easy testing of chat and email feature, some test users are required.

13. Users confuse logging into the Facebook application with logging into Facebook. The authentication mechanism provided for Facebook in this plugin is actually for authorising the Facebook application but not for authenticating the user with Facebook platform.

8.4 Conclusion

Fully working and tested software is evaluated in this section. The evaluation is done using ISO/IEC 9126 standard, which is briefly explained in section 8.2. The sample users chosen for evaluation was a subset of users selected for a survey in requirements phase. The selected set of users were both ST and Facebook users and hence they were aware of both the systems. The feedback from them is used to evaluate the product against the characteristics and sub-charateristics provided by ISO/IEC 9126 standard. At the end of this phase, the software is functional, tested and evaluated.

From the evaluation, it is discovered that the software contains some flaws, of which some were critical for user satisfaction and some were minor. The explanation in such cases was provided in the section 8.3.2. The improvements, that can be made to cover the flaws discovered, are listed in the chapter 9.

Chapter 9

Conclusions & Future Work

9.1 Conclusions

The software delivered, as part of this thesis, is a Sametime plugin developed using Eclipse. This plugin is developed to use with Sametime 8.0.1 client. The project started with a literature review which supported the objective of the project, to integrate IM and SN systems. A suitable development methodology was then identified and explained. Requirements of the project were analysed using a small survey conducted with a set of questions framed, whose intention was to reveal the extent of interest that home and work users express about the idea of the project. This phase helped to develop the project's business and system analysis models.

Due to time constraints, it was decided to develop implementation module for one social network (Facebook) and so the design modules for Facebook were developed and implemented along with the integration framework. Thus developed software was tested based on the guidelines provided by the IEEE standard 829. The test cases were framed to test the functionality of the software and determine if the requirements were met. A subset of sample users (who are both ST and Facebook users) selected for the requirements survey were asked to use the plugin developed and their feedback was collected to evaluate the software. ISO/IEC 9126 standard was followed for evaluation. During testing, some flaws were discovered and were also surfaced from user feedbacks. Some of these were analysed as critical for user satisfaction. This list include the Facebook user status was not refreshed in realtime, chat interface for Facebook users was not same as ST users. There were also some minor flaws including some Facebook user attributes being dropped while integration and at the same time some ST person attributes were not used. The reasons for these flaws not being fixed were also explained.

9.1.1 Success of the software

The success of the software when analysed using the aims and objectives as described in section 1.3 is as below

- **Objectives of the project:**
  - *To integrate enterprise messaging and collaboration with social networks:* The plugin is tested for the integration between Sametime which is an enterprise messaging system and Facebook which is a social networking system. So this objective is attained.
  - *To deliver the solution as a desktop tool:* Sametime is a desktop tool and the software is a Sametime plugin so this objective is also attained.

- **Aims of the project:**
  - *To use an instant messaging system for communications in organisational network:* IBM Sametime is used which is an enterprise Instant Messaging system.
  - *To provide uniform interface for compatible social networks:* The integration module provides a common framework which can be implemented for different Social Networks.
  - *The UI (User Interface) to pretend as the UI of Organisational instant messaging system for social network contact lists:* The UI of Sametime is used for all Social Networks and so it appears to be same for both organisational contacts and social network contact lists.
  - *To provide as many services of social networks as possible; this is limited by the scope of API (Application Programming Interfaces) exposed by social networks which is an external constraint:* The features interested to the sample users (home and work users selected for the survey in requirements phase) were implemented in the plugin developed due to time constraints.

As per the above discussion, the project is a success also as per the feedback from the evaluation survey.

9.2 Future Work

The evaluation results help the process of improving the product and the improvements are summarised in this section.

- **Using a server component**

  As the call back mechanism is not possible for a pure client project as this, other possibilities to install a server component, that takes care of call back mechanisms for Facebook, can be explored. The server component should be responsible for registering the events with Facebook and when it receives any event notifications, it should notify the clients about them. This is useful to solve the problems related to automatic status updates and automatic session management.

- **Using Ontologies**

  To avoid implementation of each of the social network, for integration, one more level of commonality can be introduced. This level is to implement a common ontology in the common API and do all the integration in the common API itself. But this again requires the implementation modules to conform the ontology of the social network to the common ontology. This approach proposes a totally new architecture for the project so is not exactly an improvement.

- **UI improvements**
  - Preference pages can be more organised. A common preference page for the project which contains the set of preference pages, one for each implementation module installed.

- Current implementation for the container of users from one SN is a group in buddy list. But in case of performing any group actions, the users moved from one group to another will not be tracked. So the application should actually parse all members of all groups in buddy list, for the users of that particular SN (which might not be an effective solution). Another way is to get users from user manager of that SN.

- **Feature improvements**
  - The application should also allow a way to browse and add friends of friends which is one of the main features of social networks.
  - The application adds all the friends of the user as a flat list to a single group, dedicated for that SN. But it can add the user's friends in the same hierarchy as they are stored in the SN website.

Bibliography

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002), 'Agile software development methods', *Review and analysis* .

Ackerman, M. S. (1998), 'Augmenting organizational memory: a field study of answer garden', *ACM Trans. Inf. Syst.* **16**(3), 203–224.

Booch, G. (1998), 'Leaving Kansas', *IEEE Software,* **15**(1), 32–35.

Borch, O. (1994), 'The process of relational contracting: developing trust-based strategic alliances among small business enterprises', *Advances in Strategic Management* **10**, 113–135.

Bradac, M., Perry, D. & Votta, L. (1993), 'Prototyping a process monitoring experiment', *Software Engineering, 1993. Proceedings., 15th International Conference on* pp. 155–165.

Burt, R. (1992), *Structural Holes: The Social Structure of Competition.*

Chua, B. & Dyson, L. (2004), Applying the ISO 9126 model to the evaluation of an e-learning system, *in* 'Beyond the Comfort Zone: Proceedings of the 21 stASCILITE Conference', pp. 184–190.

Churchill, E. & Halverson, C. (2005), 'Guest editor's introduction: Social networks and social networking', *Internet Computing, IEEE* **9**(5), 14–19.

Cross, R., Borgatti, S. & Parker, A. (2002), 'Making Invisible Work Visible: Using Social Network Analysis to Support Strategic Collaboration', *CALIFORNIA MANAGEMENT REVIEW* **44**(2), 25–46.

Cross, R., Nohria, N. & Parker, A. (2002), 'Six Myths About Informal Networks-and How To Overcome Them', *MIT Sloan Management Review* **43**(3), 67–76.

Cummings, J. N., Butler, B. & Kraut, R. (2002), 'The quality of online social relationships', *Commun. ACM* **45**(7), 103–108.

Ehrlich, K. & Cash, D. (1999), 'The Invisible World of Intermediaries: A Cautionary Tale', *Computer Supported Cooperative Work* **8**(1-2), 147–167.

Facebook Developers Wiki (2008*a*), 'Error codes - facebook developers wiki', viewed 14 October 2008, http://wiki.developers.facebook.com/index.php/Error_code.

Facebook Developers Wiki (2008*b*), 'Using batching api', viewed 20 September 2008, http://wiki.developers.facebook.com/index.php/Using_batching_API.

Facebook Press Room (n.d.), 'Facebook statistics', viewed 10 November 2008, http://www.facebook.com/press/info.php?statistics.

Forbes (2007), 'Popular social networks based on the number of active users', viewed 1 November 2008, http://www.forbes.com/2007/08/22/social-nets-marketing-oped-cx_ekc_0823social_slide_2.html.

Fowler, M (2000), 'The new methodology', viewed 12 September 2008, http://martinfowler.com/articles/newMethodology.html.

George Chin, J., Myers, J. & Hoyt, D. (2002), 'Social networks in the virtual science laboratory', *Commun. ACM* **45**(8), 87–92.

Google Code (2008), 'Revision 460: /tags/release-1.8.0/facebook-java-api', viewed 30 November 2008, http://facebook-java-api.googlecode.com/svn/tags/release-1.8.0/facebook-java-api/.

Google Code (n.d.*a*), 'Hosting opensocial apps', viewed 1 October 2008, http://code.google.com/apis/opensocial/container.html.

Google Code (n.d.*b*), 'Who3́9;s using it?', viewed 1 October 2008, http://code.google.com/apis/opensocial/whoisusingit.html.

Google Code: Issues (2008), 'Issue 124 - facebook-java-api', viewed 24 October 2008, http://code.google.com/p/facebook-java-api/issues/detail?id=124.

Greve, A. & Salaff, J. (2003), 'Social Networks and Entrepreneurship', *Entrepreneurship Theory and Practice* **28**(1), 1–22.

Hampton, K. & Wellman, B. (2003), 'Neighboring in Netville: How the Internet Supports Community and Social Capital in a Wired Suburb', *City and Community* **2**(4), 277–311.

Handcock, M. (2002), Assessing degeneracy in statistical models of social networks, *in* 'Workshop on Dynamic Social Network Analysis, Washington, DC, November'.

Handel, M. & Herbsleb, J. D. (2002), What is chat doing in the workplace?, *in* 'CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work', ACM, New York, NY, USA, pp. 1–10.

Hansen, E. (1995), 'Entrepreneurial network and new organization growth', *Entrepreneurship Theory & Practice* **19**(4), 7–19.

Herbsleb, J. D., Atkins, D. L., Boyer, D. G., Handel, M. & Finholt, T. A. (2002), Introducing instant messaging and chat in the workplace, *in* 'CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 171–178.

Hi5 (2008), 'Opensocial 0.8 in beta on hi5', viewed 26 September 2008, http://www.hi5networks.com/developer/2008/08/opensocial08inbetaontheh.html.

Hoff, P., Raftery, A. & Handcock, M. (2002), 'Latent Space Approaches to Social Network Analysis', *Journal of the American Statistical Association* **97**(460), 1090–1098.

IBM Lotus Sametime (n.d.), 'Ibm software - ibm lotus sametime', viewed 3 December 2008, http://www-01.ibm.com/software/lotus/sametime.

IBM Press room (2006), 'Ibm ships lotus sametime 7.5, delivering the first unified platform for business collaboration', viewed 30 November 2008, http://www-03.ibm.com/press/us/en/pressrelease/20259.wss.

IBM Sametime Features (n.d.), 'Ibm lotus sametime - features and benefits', viewed 10 November 2008, http://www-306.ibm.com/software/lotus/products/sametime/features.html.

IEEE Standard for Software and System Test Documentation (2008), *IEEE Std 829-2008* pp. 1–118.

ISO (1986), 'ISO 8402 quality vocabulary, ISO copyright office, geneva, switzerland'.

ISO/IEC (2001), 'ISO/IEC 9126-1 software engineering - product quality - part 1: Quality model, ISO copyright office, geneva, switzerland'.

ISO/IEC (2002*a*), 'ISO/IEC 9126-2 software engineering - product quality - part2: External metrics, ISO copyright office, geneva, switzerland'.

ISO/IEC (2002*b*), 'ISO/IEC 9126-3 software engineering - product quality - part3: Internal metrics, ISO copyright office, geneva, switzerland'.

ISO/IEC (2002*c*), 'ISO/IEC 9126-4 software engineering - product quality - part4: Quality in use metrics, ISO copyright office, geneva, switzerland'.

Jamali, M. & Abolhassani, H. (2006), Different aspects of social network analysis, *in* 'WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence', IEEE Computer Society, Washington, DC, USA, pp. 66–72.

Kautz, H., Selman, B. & Shah, M. (1997), 'Referral web: combining social networks and collaborative filtering', *Commun. ACM* **40**(3), 63–65.

Keyes, J. (2003), *Software Engineering Handbook*, Auerbach Pub.

Kruchten, P. (2003), *The Rational Unified Process: An Introduction*, Addison-Wesley Professional.

Lam, S. (n.d.), Representation of Online Social Networks, Master's thesis.

Larson, A. & Starr, J. (1993), 'A network model of organizational formation', *Entrepreneurship Theory & Practice* **17**(2), 5–15.

McDonald, D. W. (2003), Recommending collaboration with social networks: a comparative evaluation, *in* 'CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 593–600.

Mitchell-Wong, J., Kowalczyk, R., Roshelova, A., Joy, B. & Tsai, H. (2007), 'Opensocial: From social networks to social ecosystem', *Digital EcoSystems and Technologies Conference, 2007. DEST '07. Inaugural IEEE-IES* pp. 361–366.

Nardi, B. A., Whittaker, S. & Bradner, E. (2000), Interaction and outeraction: instant messaging in action, *in* 'CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work', ACM, New York, NY, USA, pp. 79–88.

Nardi, B. & O'Day, V. (1999), *Information Ecologies: Using Technology With Heart*, MIT Press.

Nielson, S., Juergensen, V., Menashes, R., Patton, T. & Schejter, M. (2002), 'Working with the sametime client toolkits', viewed 17 October 2008, http://www.redbooks.ibm.com/abstracts/sg246666.html?Open.

Ofcom, Office of Communications (2008), 'Social networking, a quantitative and qualitative research report into attitudes, behaviours and use', viewed 13 October 2008, http://www.ofcom.org.uk/advice/media_literacy/medlitpub/medlitpubrss/socialnetworking/report.pdf.

Ott, L. (2008), 'A tour of the ibm lotus sametime v7.5 toolkits', viewed 10 November 2008, http://www.ibm.com/developerworks/lotus/library/sametime75toolkits/.

Poe, R. (2001), 'Instant messaging goes to work', *Business2. 0, July* .

Preece, J., Maloney-Krichmar, D. & Abras, C. (2003), 'History of Emergence of Online Communities', *Encyclopedia of Community. Berkshire Publishing Group, Sage* .

Project, P.I.A.L (2002), 'Getting serious online: As americans gain experience, they use the web more at work, write e-mail with more significant content, perform more online transactions, and pursue more serious activity', viewed 11 October 2008, http://www.pewinternet.org/reports/poc.asp?Report=55.

Punter, T., Van Solingen, R. & Trienekens, J. (1997), Software Product Evaluation, *in* 'Proceedings of 4th European Conference on Evaluation of Information Technology (EVIT'97), Delft'.

Quan-Haase, A., Cothrel, J. & Wellman, B. (2005), 'Instant Messaging for Collaboration: A Case Study of a High-Tech Firm', *Journal of Computer-Mediated Communication* **10**(4).

Reynolds, P. (1991), 'Sociology and Entrepreneurship: Concepts and Contributions', *Entrepreneurship Theory and Practice* **16**(2), 47–70.

Rittinghouse, J. & Ransome, J. (2005), *IM Instant Messaging Security*, Digital Press,Newton, MA, USA.

Rosenthal, E. (1997), 'Social networks and team performance', *Team Performance Management* **3**(4), 288.

Smyth, P. (2003), Statistical modeling of graph and network data, *in* 'IJCAI Workshop on Learning Statistical Models from Relational Data'.

Starr, J. & MacMillan, I. (1990), 'Resource cooptation via social contracting: Resource acquisition strategies for new ventures', *Strategic Management Journal* **11**(4), 79–92.

Sundén, J. (2003), *Material virtualities: approaching online textual embodiment*, P. Lang.

Van Laere, K. & Heene, A. (2003), 'Social networks as a source of competitive advantage for the firm', *The Journal of Workplace Learning* **15**(6), 248–258.

Weaver, A. & Morrison, B. (2008), 'Social networking', *Computer* **41**(2), 97–100.

Wellman, B. & Gulia, M. (1999), 'Net-Surfers Don't Ride Alone: Virtual Communities as Communities', *Networks in the Global Village: Life in Contemporary Communities* pp. 331–66.

Wells, D. (2001), 'Extreme Programming: A Gentle Introduction', viewed 20 September 2008, http://www. extremeprogramming. org.

Appendix-A

Figure 1is a transcript of chat with Jason Simoneau (IBM), ST expert in SDK area.

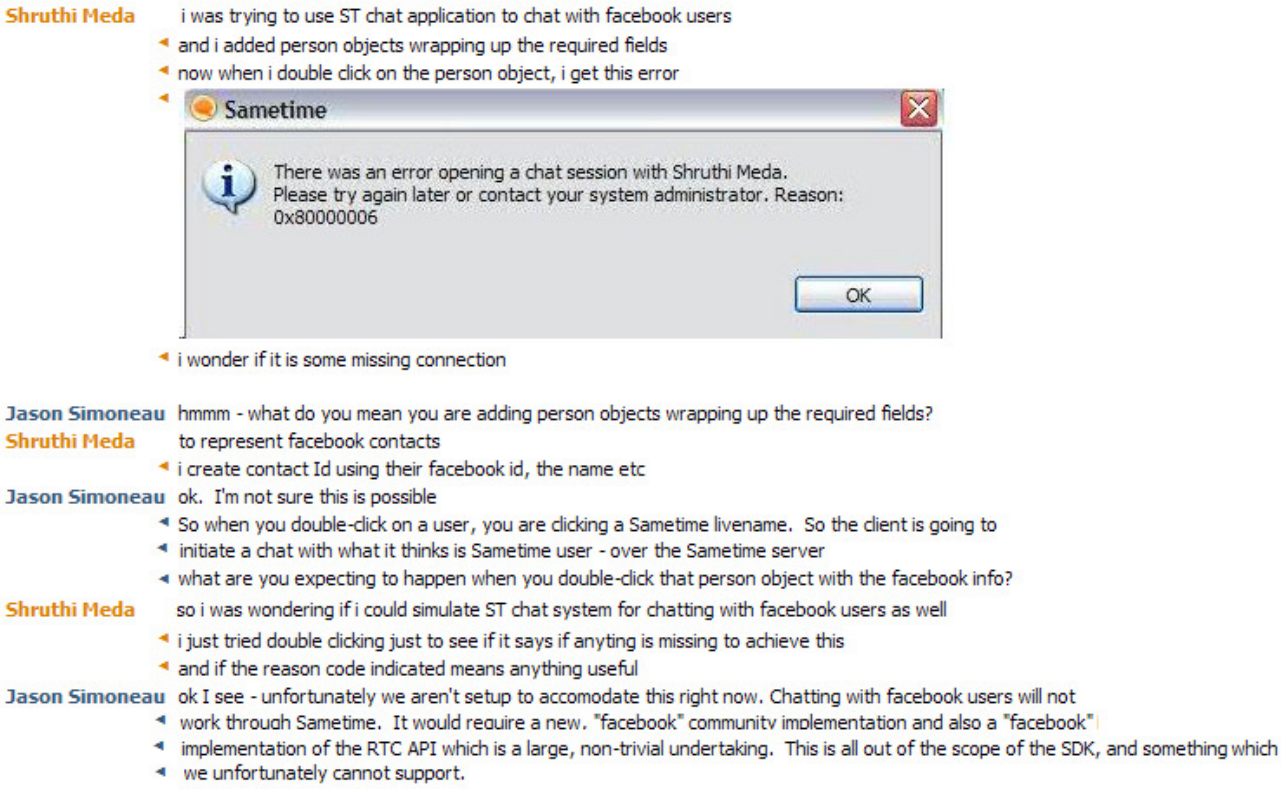


Figure 1: Chat with Jason Simoneau

Appendix-B

The survey questions used for requirements analysis is shown in the figure 3 and the notes shown in figure 2 to assist users in finding what types of SN users they are.

Notes Before filling the questionnaire:

According to Ofcom's qualitative research, social networkers could be differentiated into five groups based on the difference in their attitude to the social networking sites and their behaviour while using them. These groups are described as follows.

• **Alpha Socialisers**: The people who use the social networking sites very often for flirting, meeting new people. Their main motive is being entertained.

• **Faithfuls:** The people who used the social sites to maintain their old friendships generally from their school or universities.

• **Attention Seekers:** The people who used the social sites to seek the attention from people and were so keen on obtaining comments from others about their profiles or photos uploaded or anything published about them.

• **Followers:** The people who joined the social sites just because others were doing or to keep up with the changing technologies or to keep up with their peers.

• **Functionals:** The people who joined the social sites for specific purpose.

While the majority of internet users were part of Social networking, there were people who were not using the the sites for varied reasons. The report also classified the non-users of social networking based on the reason for not using them as follows:

• **Concerned about safety:** Many people were afraid of using the social sites because they were afraid of publishing their personal details on the internet. They were concerned about the safety of making their details available online although technically they would not need to publish legitimate or correct details.

• **Technically inexperienced**: There were a portion of non users of social sites who were not aware of using the technology and who lack the confidence in using computers and the internet.

• **Intellectual Rejecters:** These type of people would not have any interest in using these sites at all and see using them as just a waste of time.

Figure 2: Notes given as part of survey

Questionnaire about integrating IM(Instant Messaging) and SN (Social Network) systems into a single application

1. Do you use any of the IM or SN systems? Select relevant option.
   - ☐ Neither of the systems
   - ☐ Either of the systems
   - ☐ Both the systems
2. If using only one of those, would you be interested in using the other system?
   - ☐ Yes
   - ☐ No
3. If already using or interested in using both the systems, would you like to use them in a single interface?
   - ☐ Yes
   - ☐ No
   - ☐ Does not matter
4. Are you a home user or corporate user (use them at work)?
   - ☐ Home
   - ☐ Corporate
5. Based on the notes given in the document, what type(s) of user are you?

Please answer the following questions if your answer is "Yes" for Question 3.

6. Would you like to use them as a desktop application or as an Internet application and why?
7. What Social Networks do you want to be integrated in to your IM?
8. What features of SN would you like to be integrated into IM as a must?

Figure 3: Questionnaire used for requirements analysis

Appendix-C

Facebook Java toolkit license is shown in the figure 4.

```
+-----------------------------------------------------------------------------+
| Facebook Development Platform Java Client                                   |
+-----------------------------------------------------------------------------+
| Copyright (c) 2007 Facebook, Inc.                                           |
| All rights reserved.                                                        |
|                                                                             |
| Redistribution and use in source and binary forms, with or without          |
| modification, are permitted provided that the following conditions          |
| are met:                                                                    |
|                                                                             |
| 1. Redistributions of source code must retain the above copyright           |
|    notice, this list of conditions and the following disclaimer.            |
| 2. Redistributions in binary form must reproduce the above copyright        |
|    notice, this list of conditions and the following disclaimer in the      |
|    documentation and/or other materials provided with the distribution.     |
|                                                                             |
| THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR        |
| IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES   |
| OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.     |
| IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,            |
| INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT    |
| NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,   |
| DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY       |
| THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT         |
| (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF    |
| THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.           |
+-----------------------------------------------------------------------------+
| For help with this library, contact developers-help@facebook.com            |
+-----------------------------------------------------------------------------+
```

Figure 4: Facebook Java toolkit license

Appendix-D

Facebook application created for the purpose of this project is shown in the figure 5.

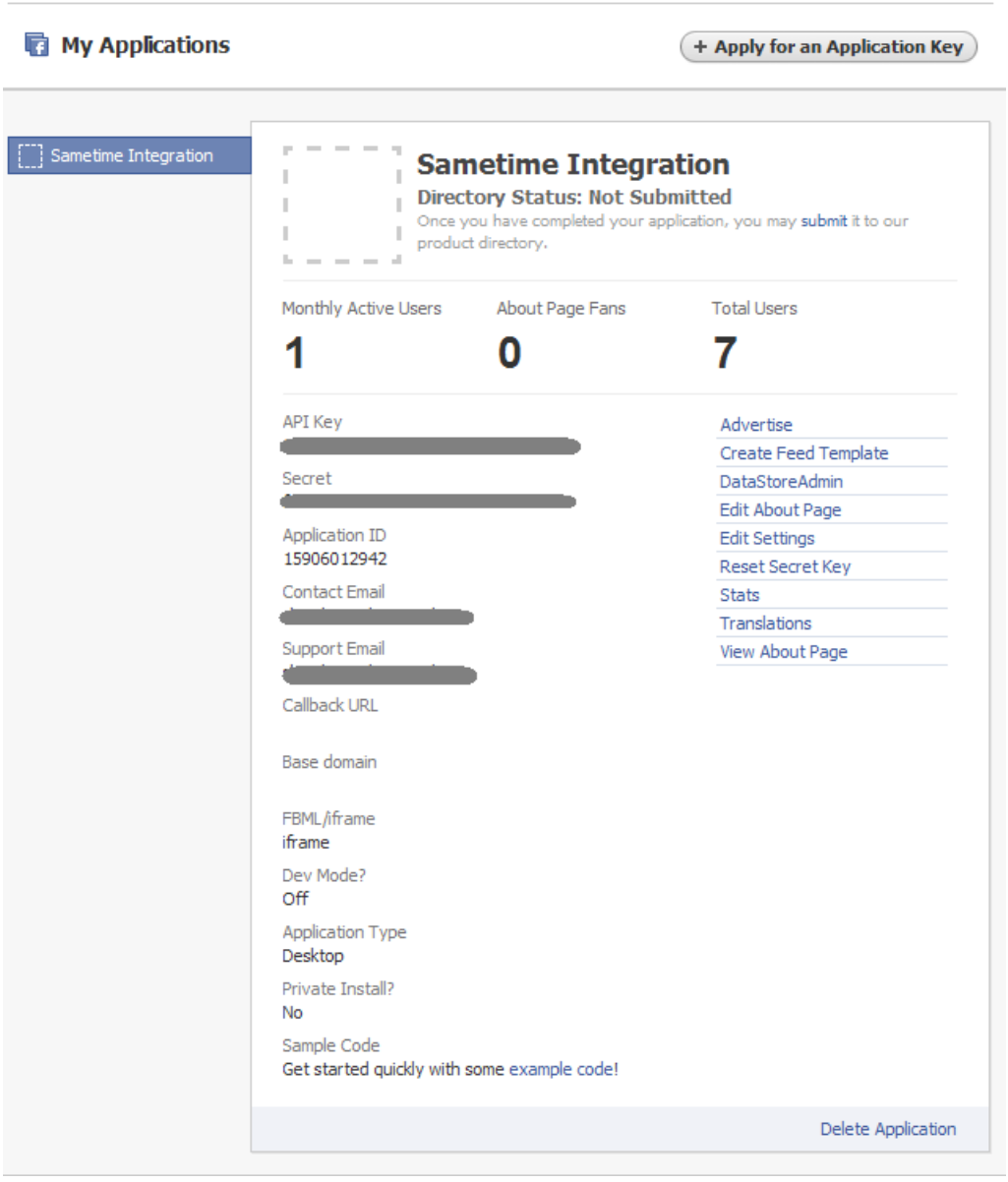


Figure 5: Facebook application details page

Appendix-E

The mail thread with a facebook developer regarding facebook chat problem.

LiveMessage.Send problem Inbox | x

Shruthi Meda to andrhahn

Hi Andrhahn,
Sorry for a private mail but i am really stuck with using LiveMessage.Send() in my desktop application.
Am doing a proof of concept for my college project and have a very near deadline.
I raised a ticket http://code.google.com/p/facebook-java-api/issues/detail?id=124.
Can you please help me using this API.
I used this API :

```
client.useBetaApiServer();
JSONObject sendMsg = new JSONObject();

sendMsg.put("from", <logged in user id>);
sendMsg.put("msg", "text to send as live message");
boolean sent = client.liveMessage_send(new Long(uid), "chatEvent"+uid,sendMsg);
```

and get this error:

```
<?xml version="1.0" encoding="UTF-8"?>
<error_response xmlns="http://api.facebook.com/1.0/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
  <error_code>0</error_code>
  <error_msg>Success</error_msg>
```

Andy to me

Shruthi,

This isn't working because there is a bug in the code. I will add a new issue for this.

Instead of the POST passing 'uid', it should be passing 'recipient', as in http://wiki.developers.facebook.com/index.php/LiveMessage.send.

POST: http://api.new.facebook.com/restserver.php?
api_key=xxxxx
call_id=3454354350779&
event_name=chatEvent732076781&
format=xml&
message=%7B%22
from%22%3A732076781%2C%22
msg%22%3A%22
text+to+send+as+live+message%22%7D&
method=facebook.livemessage.send&
session_key=xxxxx
sig=k8c908sad90sa8d87225c803ce6ba2&
uid=938371573&v=1.0

Andy