UNIVERSITY OF GOTHENBURG

Social-Aware Applications

Study on Social Context Aware Applications: Exploring Potential Usages, Defining Requirements and Understanding Issues

Master of Science Thesis in Computer Science

GÖKHAN BERBEROĞLU

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, November 2011

Social-Aware Applications
Study on Social Context Aware Applications: Exploring Potential Usages, Defining Requirements and Understanding Issues

GÖKHAN BERBEROĞLU (gberberoglu **at** yahoo.com)

Examiner: MARINA PAPATRIANTAFILOU (ptrianta **at** chalmers.se)
GEORGIOS GEORGIADIS (georgiog **at** chalmers.se)

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden November 2011

Preface

Age of social networks and mobile applications is upon us. I believe we passed "web 2.0" and entered a new mobile-social era. While being fascinated by every aspect of social networks and ingenious mobile apps, I usually get angry to myself about being too lazy to produce something instead of just consuming. While having strange ideas, I was lacking of motivation and guidance. Finally, with SenseMe, I had a chance to contribute something in this area.

It is fascinating to see how a simple idea as "what if I share my Facebook id with my phone" turns into a complex, all around mobile social networking system. For this, I want to thank my supervisors Marina and Georgios. Their experience and ideas have made a simple idea to grow into a solid thesis.

I want to thank all my friends in Gothenburg, especially Yağız Mungan and Serkan Karakış, with whom I discussed every detail, used as pre-alpha testers or proofers.

Finally yet importantly, I really appreciate my father Serol and my mother Neşe for their support made it possible to pursue a master's degree in University of Gothenburg.

Abstract

Social Networking Sites are becoming more and more important on people's daily lives. Combined with the mobility of the user, social data bring unique opportunities and creates a new kind of "mobile social" applications. In this thesis, social awareness of mobile applications is investigated on both social and technical aspects including privacy. Along with the definition of social-awareness and privacy rules, a mobile social application called SenseMe is presented that adds social layer to opportunistic networks and converts regular mobile phones to social agents. The prototype application is implemented using JavaMe and peer-to-peer aspects were validated in a real life experiment, which shows that social-awareness does not requires powerful hardware with fully featured operating systems.

Keywords: social awareness, opportunistic networks, mobile social networks, (Bluetooth)

Table of Contents

Table of Figures

Chapter 1 – Introduction

1.1 Thesis Background

The Growth of the social network sites (SNS) is exceeding predictions. The World's most popular social networking site Facebook started 2009 with 150 million users, whose founder considered as a "milestone". By summer 2010, Facebook has 500 million users and 150 million of them are actively using mobile applications to connect [1].

SNSs contain enormous volumes of social context information related to users. Social context information including friendship relations between users, event attendance, favorite books, movies etc. are already heavily used for displaying customized advertisements on the web pages. On the other hand, unfortunately most of the mobile social applications are just simplified user interfaces adapted for specific mobile devices. The opportunities for the mobile applications that can make us of this data will be investigated in *2.5 Social Aware Application Examples*.

The motivation behind this thesis is that in the near future social-aware applications will be more popular, running in mobile phones, car audio systems, billboards at streets etc. Using mobile phones as an agent for social context, users will be able to interact socially with each other and with other "smart" applications. For example, while sitting in a café, user's mobile phone can find someone who attends the same event (peer-to-peer) or when he/she goes to a bookstore and approach the kiosk, it can recognize the user and suggest books according to his/her profile (peer-to-service).

Social integration and awareness are hot topics for both industry and academia. There are already some commercial mobile applications available like "LoKast" on iPhone that allows people to share media with nearby users with social profile support [2] and many academic studies exist, which are investigated in *2.6 Other Works Related With SenseMe.*

The opportunities in introducing social profile information located on the web to the real life meetings are huge. In this thesis, a framework and prototype application called SenseMe are presented to benefit the potential. The proposed prototype brings social network relations to real-life meetings, helps users discover people with similar tastes and allows third party applications (referred to “services” later) to serve customized content. It is designed with respect to privacy, while allowing the user to share information and discover new friends in a safe ecosystem. In contrast to other presented solutions, SenseMe is capable of running on ordinary JavaMe enabled mobile phones while requiring minimum internet connectivity and respecting privacy on a distributed manner.

1.2 Purpose of the Thesis

The purpose of the thesis is to study social context aware applications by exploring potential usages, defining requirements and understanding issues. The study investigates requirements, issues and limitations to enable social awareness on mobile applications. Examples of application types are explored and different communication techniques are compared.

As an outcome of the thesis, based on the research, a framework including file format to carry user’s social profile and a communication protocol to exchange this data are introduced. For demonstrating the capabilities of the concept and usability of the framework, a prototype mobile application “SenseMe” is developed.

1.3 Methodology

In order to form a solid theoretical background study on previous publications about mobile social networks are completed. Initial study includes privacy aspects, wireless communication protocols and users’ social behaviors on online networks. In addition, many professional applications and websites are investigated. After understanding the concept and current solutions, requirements and specifications for a new framework to overcome current limitations are defined. Using the research file format, protocol and system architecture are designed and prototype SenseMe application is implemented. Finally, real user experiments of the prototype application are conducted and results are discussed.

Chapter 2 – Social Aware Applications

This chapter contains background study about Social Networks, Context Awareness, introduction to required concepts such as computer networks types, technical and privacy related issues. Although the thesis focuses on mobile applications, many of the findings can be applicable to any kind of applications like websites. Outcomes of the study are specifications and requirements for SenseMe and used in designing the system. In the report, the term “Social Context Aware” is sometimes shortened as “Social Aware”.

2.1 Social Networks

A “Social network” can be defined as a collection of nodes that are connected through some kind of relation or relations. Relations may be based on domains like educational, professional etc. For example, in professional domain a social network can be formed among people as “common company” relation, between the colleagues.

Boyd & Nelson [3] define “Social Network Sites” with the following properties:

“web-based services that allow individuals to construct a public or semi-public profile within a bounded system, articulate a list of other users with whom they share a connection, and view and traverse their list of connections and those made by others within the system” which will be used through the thesis.

The History of SNSs backs to 1997 with the launch of SixDegrees.com website. It is named after “six degrees of separation” theory, presented by Frigyes Karinthy [4], a Hungarian author and journalist. It refers that any random two people around the world can be connected by at most six steps (people): chain of friend-of-friend. This theory or concept was investigated by many companies like Facebook or Microsoft and surprisingly found reasonable. Microsoft Research studied 30 billion instant messages sent in one month in 2006 and according to their studies the chain can be constructed with seven people [5]. Later, more popular sites like Friendster, Facebook, MySpace and LinkedIn emerged with attracting millions of users.

Since SNSs grow to be a huge market, functionality becomes more and more complex. What made SixDegrees.com special were simple functionalities like social profile creation, friend lists and personal messages. Together with the web 2.0 thrust, SNSs have changed and improved so much compared to previous friendship sites. New generation SNSs not only contain advanced functionalities like photo tagging or "like" feature (will be explained later); but also provide APIs for third party developers to integrate. This is one of the main reasons why Facebook separated from the others. First, they provided tools to develop Facebook applications that integrate the website with blending UI. Later, they decided to allow third party websites and programs to communicate with FB and launched Facebook Connect in 2008.

From social-awareness (will be defined in *2.2 Context Awareness*) point of view, stored data on these sites are very important. Typical data stored about user includes:

- Personal information
- Location info
- Work and education history
- Friend lists

Types of data stored on modern SNSs are also changed. In addition to social data itself, relation information is stored now. For example, a user can tag other users on a photo, which converts a static object like photo into a relational object. In addition, by integrating third party websites, SNSs can collect social information from user's social interactions. For instance, Facebook provides "like" feature integration with websites. Users can click a "like" button for an article on a game review site, which will immediately appear on users profile page as news feed. More recent trend is location check-ins; users with GPS capable mobile devices submit their locations to the SNS (Figure 1).

So now, in Facebook example, additional data stored about user includes:

- Friends in photos by tagging feature
- User's behavior / social preferences in third party websites
- Users location history

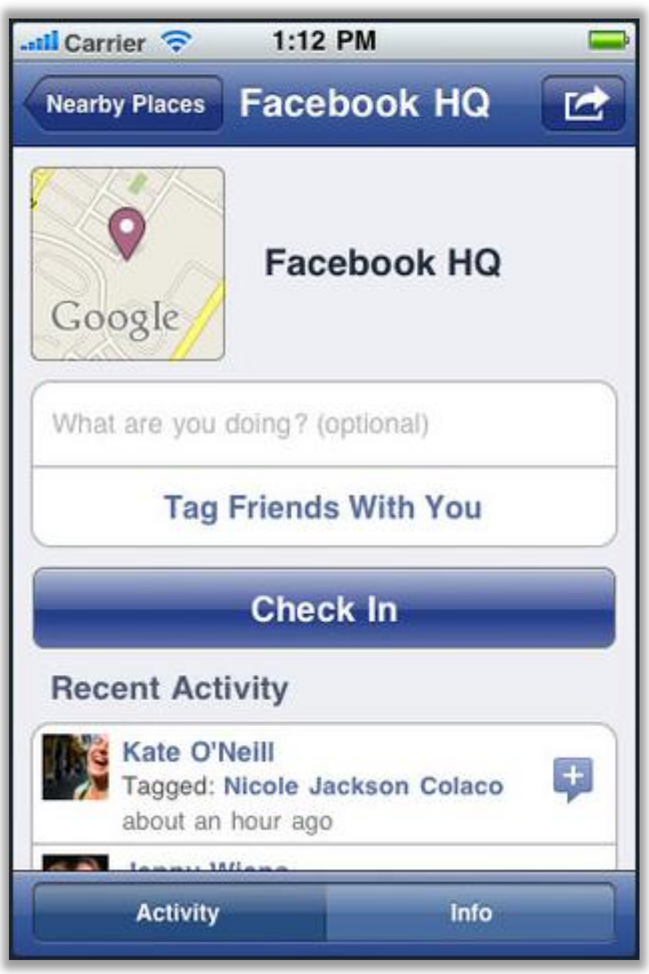


Figure 1 – Facebook Places Location Check-in (©Facebook)

All the information once required huge efforts to collect is available online now. Interesting part is that users willingly give the information themselves, which will be covered in *2.4 Privacy and Security Issue*.

Another feature that integration APIs provide is "single registration", which became a catalyzer for industry acceptance. Web site developers do not have to implement user registration features; instead, they can simply integrate web pages with one of the leading SNS and get almost all the information covered above.

Proposed solution, SenseMe, relies heavily on SNSs. In order to avoid duplicate information and putting tedious tasks to the user, SenseMe fetches user's data directly from social network site. There are many advantages of this approach compared to direct user input. First of all, users are reluctant to fill another registration form and skeptical about giving private information. SenseMe or another web page cannot provide same feeling of “trust” and “security” like Facebook or Twitter. In addition, there is no way to get certain kind of data directly from user input, like Friends or Events. Therefore, with the integration, SenseMe can gather huge amount of social data about user.

Understanding user interaction i.e. how they use social network sites, how they manage relations is vital for social-aware application developers. Previous studies related interaction can be grouped as online and offline aspects. Online aspects include user identity management [6] and grouping behavior [7]. Offline aspects are more important for third party applications point of view; since they focus on real world face-to-face usage of SNSs. Barkhuus & Tashiro on their study on students using Facebook, found that Facebook heavily used to facilitate ad-hoc social gathering by events and simple profile wall posts. Another important analysis presented is people avoid invitation / dialogue via phone, but prefers sending a personal message or putting a general wall-post on Facebook [8].

In conclusion, social network sites contain tremendous amount of information that can be used to enhance user experience on both online and real-life, for the applications that are smart enough to use them.

2.2 Context Awareness

In order to understand the concept of this thesis, the terms “context” and “context awareness” should be defined.

Context is all the conditions and circumstances that are relevant to an entity. From the computer science perspective, there are slightly specialized descriptions. Schilit & Adams et al. implicitly defined contexts as location, collection of nearby objects and changes to such things over time [9]. More recent and applicable definition used in this thesis comes from D. Key: *“Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.”* [10]. It is clear that even though the definition remains the same, the contents of a context depend on the domain. The context used in this thesis refers to social domain and contains all the social information.

Social context contains all the information listed in *2.1 Social Networks* also can contain sensory data like the temperature or lightning of the room, nearby users, location information. As the technology advances, algorithms that are more complex are being developed to interpret static data and create context information. An interesting study on this area is CenceMe [11] (naming resemblance with SenseMe is purely coincidental) by Miluzzo & Lane et al. can interpret sensory data to create presence information. For example, by interpreting mobile phones motion sensor CenceMe can select presence i.e. "running", "sitting", by monitoring the device microphone it can understand if user is having a conversation. An actual working application presented as "Your Seventh Sense" for Apple iPhone & iPod Touch is available in cenceme.org. There are too many data stored on mobile phone to process and create social information. For instance, by examining user's call and SMS history another list of close friends can be defined.

Context awareness is the ability to understand, adapt and use the context information. We are as humans naturally context aware. In our daily lives, we can sense the context and adapt to it. For instance, in a dialogue about a movie, we can understand other person's movie taste and recommend him/her movies. Computers on the other hand currently are not able to do these complex tasks automatically. The protocol and file format presented in this thesis aims to solve first part of this problem, providing the context information. Context awareness from the application perspective term first discussed by Schilit & Theimer *"the ability of a mobile user's applications to discover and react to changes in the environment they are situated in"* [12] . Since this study is about social context related applications, a more specific definition of [10] is presented: *"A system is social context aware if it uses social data to customize its services*".

2.3 Technical Issues

Getting the social data is discussed in *2.1 Social Networks*; this chapter discusses the remaining questions; why and how to share the context data for mobile phones.

2.3.1 Opportunistic Networks

People behavior while moving around with wireless devices and meeting with other people can be best modeled by **opportunistic networks (ON)** (Figure 2). ONs are also called **Delay Tolerant Networks (DTN)** as there is no common agreed terminology [13]. ON is a temporary self-organized network formed by transient nodes for a short period. ONs are different types of **Mobile Ad-Hoc Networks (MANET)**, which is a type of collaborative network that requires no infrastructure and forms on the fly. Most important difference between MANETs and ONs are is clearly the **perception of mobility**. For MANETS, mobility is a challenge to overcome but for ONs, it is an opportunity for data forwarding and context sharing.

The important characteristic of ONs is that there is no guarantee that nodes will be connected in the future, like a random person passing by in the street. Another characteristic is **Delay Tolerance**, if there is a chance to meet with same node again ONs must be delay tolerant since no time estimation about meeting is possible. Early researches on ONs are mostly based on routing issues but as the mobile devices are getting more complex and feature complete, new application areas like social relations are introduced. Currently Delay Tolerant Networking Research Group of Internet Research Task Force (IRTF) studies architecture and protocols of ONs. [14]

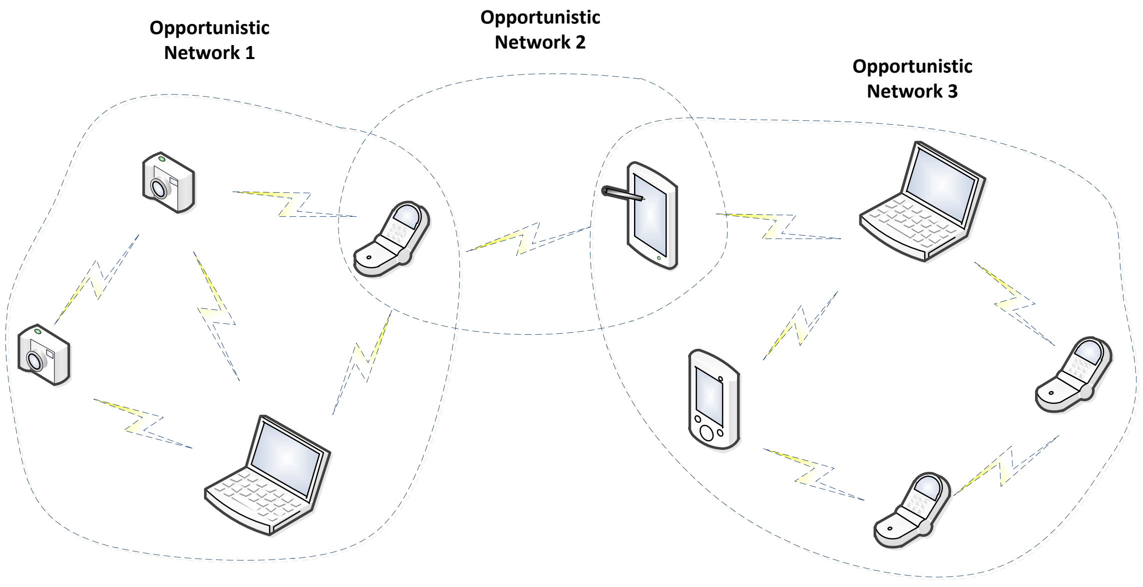


Figure 2 – Opportunistic Networks

Network formation in opportunistic networks is related to the communication technology. If mobile phone has **internet access**, then a direct connection between devices is not necessary. Using Global Positioning System (GPS) data with a centralized solution, a **location based** overlay network can be created with users that are in the same radius. In this approach, a central server constantly monitors client positions and notify clients about neighboring peers via internet thus creating a “controlled” ON. These type systems are more suitable for **Location Awareness** or **Location Based Social Networks**. Location Awareness is a popular trend right now among the SNSs, there are many commercial websites providing content like nearby travellers/restaurants etc. based on users location and major players like Facebook introduced Facebook Places to support location awareness [15]. On the other hand, **Personal Area Network (PAN)** is the mostly capable of creating and supporting ad-hoc opportunistic networking. Since PAN ranges are limited, phones can form a **physical proximity based** networks. Most adopted PAN standards in mobile phones are **Wi-Fi** and **Bluetooth**, which are covered in next section.

In the real social human life, people move and meet with new people opportunistically, where they form new friendship relations if they found other party interesting or the create trust chains for unknown people, taking common friends as references. Some people are more social than others are; they travel a lot, meet more people etc. Such properties of human life present great solutions for many problems like routing or privacy in MANETS. For instance, Bubble Rap [16] by Huri & Crowcroft et al. is a social-based forwarding algorithm in ONs. Basic idea behind Bubble Rap is using popularity of an individual (centrality) in a community to improve routing, i.e. selecting people with more friends. In addition, trust issues in node selection can be solved friendship relation instead of central authority based solutions.

In ONs, highly mobile nodes randomly meet each other and receive new kinds of information, like sensor data. By this context, ONs took advantage of human behavior instead of the traditional networking paradigms. Conti & Kumar [17] clearly defines relation between electronic social networks (ONs), human social networks (real life friends, colleagues etc.) and virtual social networks (SNSs). ONs are in the middle of human and virtual social networks and can take advantage from both of them.

2.3.2 PAN Protocols for Mobile Phones

Bluetooth is a wireless networking technology designed to allow devices for short range communicating in ad-hoc fashion. Bluetooth supports groups called “piconet” up to eight nodes, one master and seven slaves. Because of the cheap hardware, Bluetooth quickly became industry standard in mobile phones. Today, almost all mobile phones support Class 2 Bluetooth that allows effective communication up to 10 meters.

Wi-Fi is the user-friendly name of IEEE 802.11b standard. It belongs to the IEEE 802.11 set of standards for wireless local area networks (WLAN), supporting two types of WLANs: infrastructure and ad-hoc. Infrastructure mode consists of an Access Point (AP) node and associated situations forming a Basic Service Set (BSS). In Ad-Hoc mode, any node can initiate the network and form an Independent Basic Service Set (IBSS).

In comparison, Bluetooth has lower range, lower power consumption and higher deployment rate since it is supported in virtually all mobile phones. On the other hand, Wi-Fi has longer range, high power consumption and low availability since only supported by high-level smartphones [18].

2.3.3 Phone Limitations

In this section, phone limitations for social-aware applications will be discussed in hardware, software and connectivity aspects. Through the report, current generation of mobile phones will be divided into two groups: **feature phones** and **smart phones**.

Feature phones are the standard mobile phones running a proprietary operating system (OS) and provide basic phone functionality like calling, SMS, contact listing. Extended features are basic local area connectivity via Bluetooth and limited third party application support via JavaMe. Feature phone OSs are closed and no APIs provided for developers. Third party frameworks like JavaMe can only provide limited integration and functionality, since design motive of JavaMe is portability instead of exposing phone's unique capabilities. Internet connectivity is supported via GPRS or EDGE, but Wi-Fi is not common. According to the common carrier sales strategy, feature phones do not come with a contract that allows unlimited data communication. For this reason, constant internet connectivity is not financially efficient for the user. Hardware wise, devices have limited storage, memory and CPU power that make it impossible for media sharing applications.

Smart Phones are mobile phones that provide advanced software, hardware and connectivity services besides basic phone functionalities such as running a full feature OS, allowing native apps, Wi-Fi connectivity, GPS data etc. New generation of smart phones include Personal Data Assistant (PDA) features like touch screen and render PDA concept obsolete. Carriers often sell smart phones with unlimited data plan so users have constant internet connection. Hardware is similar to desktop computers in storage, memory and CPU.

For social aware application developers, decision of target platform depends on required functionality. Important factors to be considered are internet connection, input and display capacity and storage. For example, smart phone applications can simply share profile ID's to link with SNS and fetch latest information on demand, but this will highly depend on internet connection. On the other hand, feature phones or embedded recommender systems will require full context data to work offline. User interaction point of view, smart phone applications are clear winner since they can provide near PC visual experience. In conclusion, target platform has big impact on system architecture.

2.4 Privacy and Security Issues

Parallel to the increasing usage and media attention to SNSs, user's privacy concerns are getting bigger. For an application working with social data, presenting a solid privacy model is very important to be accepted. For this reason, in this section both social and technical sides of the issue will be explored: human behavior on sharing private information and Facebook's long debated handling of private data.

Privacy after social networks is a complex topic. User information on the SNS are shared with some kind of relation with public. Definition of "relation" depends on the SNS, it may be friends, friend-of-friends, colleagues etc.

User need for control over privacy attracted many studies. A study regarding social networks security is conducted by Nagy & Pecho [19] . In this paper, they found that people are giving private information effortlessly using social networks which otherwise requires much effort to obtain using other methods like phone or personal communication. They called this fact as "privacy paradox". The information leakages not only cause privacy breach but also give opportunity for advanced phishing attacks to both themselves and their friends.

Another interesting study is done by Brandimarte et al. [20] where they identified that people cannot distinguish "control over publication" and "control over access". When using social networks, users themselves add information and publish it so they have "control over publication" and feel safe. However, when someone else is responsible users feel like losing control so do not reveal information. However, in fact, in both cases they have no "control over access". Brandimarte et al. refer to this hypothesis as "illusion of control".

Like websites, mobile applications also have to provide mechanisms to manage privacy. After investigating previous studies [21] [22], here is presented **privacy rules** for mobile social aware applications. SenseMe should follow these rules in order to respect user's privacy.

1. Ability to select **what to share:** Applications should provide a way for selecting what kind of information can be shared.
2. Ability to select **whom to share:** Applications should provide a way for selecting with whom (remote user or system) information can be shared.
3. **Default Off** / **Default Deny:** Automatic functions that disclose information should be default off, confirmation dialogs should be default deny.
4. All transmission without user's consent should be **anonymous:** Any data that needs to be transferred without user's consent cannot contain identity related information.

All previous discussion was about application design decisions, which assume if everything goes as supposed. Another concern is security of sites and applications. SN sites are popular targets of adversaries using social engineering or technical attacks (hacks). Application developers should understand the risks to SNSs in order to design more secure social aware applications.

For example, one of the most popular social engineering attacks is called "profile cloning" and discussed by Leyla & Engin et al. [23]. Attack is based on establishing a friendship relation with the victim. In profile cloning, attacker creates a fake profile that impersonates a friend of victim and then sends a friendship request to the victim. With a high chance, victim will not detect duplication and grant friendship. To enhance the attack, friendship request message can be "Hey <victim_name>, I lost my other profile's password, this is the new one!" Once attacker gets friendship, he/she can access private information.

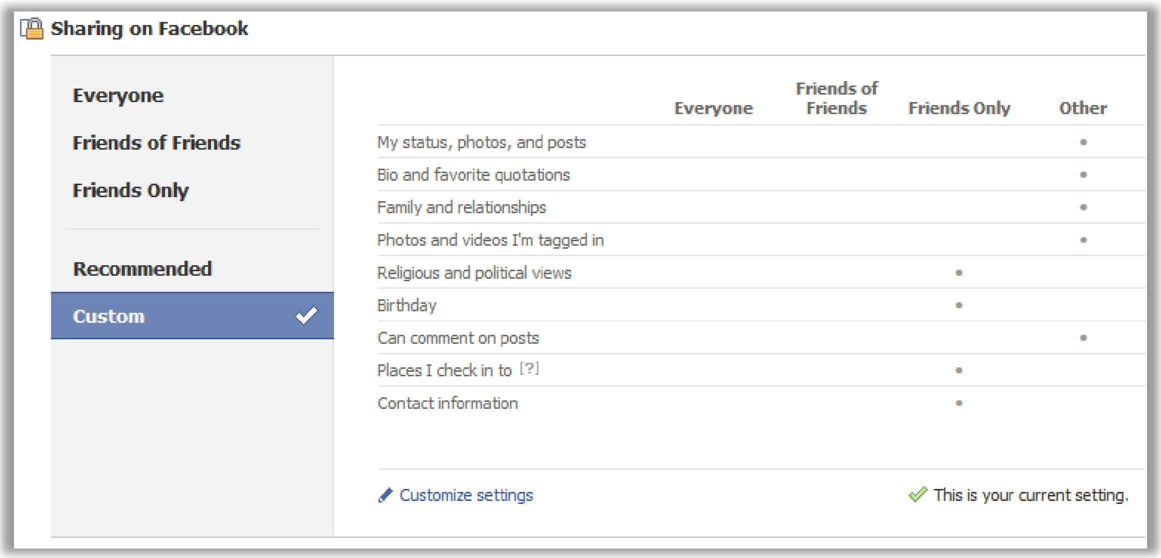
Since most of the sites also provide APIs for integrated application development, there are more fronts to cover. Integrated applications once authorized, can access most of the sensitive data depending on the platform. If the application is compromised, user's data will leak no matter how secure SNS is. That brings great responsibility for social application developers; their security is not only related with application itself but with the user's data on SNS.

2.4.1 Facebook Privacy Model and Security

Facebook stores a huge amount of data for each individual user. Of course, it can only store information that is published by the user himself/herself (or sometimes by friends), so users have ultimate control over data by simply selecting not to put it into Facebook in the first place. Facebook gives the option to select sharing data with groups, i.e. user can choose sharing a photo album with friends, friends of friends or everyone etc. (Figure 3).

The term **source** will be used as information source like *wall post*, *photo album*, *hometown information, favorites* and **sharing category** as custom, *friends only*, *friends of friends only*, *friends and networks* and *everyone*. Users can select categories for different sources.

Facebook is well known with constantly changing/updating privacy policy and mechanisms. Latest privacy update investigated in this thesis was conducted on May 2010. With the update, a new term called Publicly Available Information (PAI) is introduced that cannot be set to hidden including name, profile picture and networks. As expected, these decisions started a big debate in industry and academia [24].



	Everyone	Friends of Friends	Friends Only	Other
My status, photos, and posts				●
Bio and favorite quotations				●
Family and relationships				●
Photos and videos I'm tagged in				●
Religious and political views			●	
Birthday			●	
Can comment on posts				●
Places I check in to [?]			●	
Contact information			●	

✓ This is your current setting.

Figure 3 – Facebook Security Settings (©Facebook)

Besides the social aspects, there are technical issues also to consider. Facebook enables third party developers to develop integrated applications. To protect user's privacy Facebook enforces policies legally on applications [25]. In order to understand risks related to third party applications, a brief knowledge about technical background of integrated applications is required.

Application developers register their application with Facebook and receive a unique **Application ID** and unique **Application Secret**. When user attempts to use the application for the first time, he/she is required to authorize it. Users authorize apps by installing to their profile. Once installed, application is given permission to read a part of the user's information. Applications can request extended permissions from users such as posting to wall, offline access and even reading the inbox. Users agree basic reading permissions by installing the applications and extended permission by clicking "allow" button. Important point here is once user installed an application; data flows out of Facebook and relies on application's security. Although Facebook has policy restrictions regarding storage of information fetched, there is no way to technically enforcing it.

In the background, Facebook uses Application Secret to identify requests coming from authorized applications. So developers should keep Application Secret code secret, while they have to embed it to their application somehow to work. If somehow the secret is compromised, any adversary can impersonate the application. Many previous attacks caused private information leaks are carried by vulnerabilities in third party applications like cross-site scripting attacks.

As a conclusion, security and privacy should be integral part of the application design process. Since a system is secure as its weakest link, it does not matter how strong Facebook's security measures. They can spend millions of dollars and tens of security experts, one poorly developed 3rd party application will countermeasure all of them.

2.5 Social Aware Application Examples

Having defined context awareness and discussed related issues, here different opportunities of mobile context aware applications will be explored.

For a long time, Electronic IDs and passports are used for e-identification officially. Eventually mobile phones can be and should be the personal agents. Recent developments use NFC technology for payment via phone [26] and show that mobiles will have greater importance on human-computer relation soon. Huge deployment rate and widely supported communication technologies like Bluetooth and Wi-Fi make them perfect candidates for e-agents. One of the most important difficulties is lack of standardized application level communication protocol and data formats. In *3.2 File Format and Protocol*, a very lightweight solution will be proposed to address this problem.

Peer-to-peer applications cover user-to-user relations in this thesis and will have most benefit from context awareness particularly in highly dynamic networks (Figure 4). These types of applications usually work in an abstracted network structure, called **overlay network**. Peers select other nodes by some criteria and form an overlay network. Common example of P2P systems is file-sharing applications like Bit-Torrent. Depending on the P2P applications, peer selection criteria has big impact on system efficiency. Context information can be used as criteria in such scenarios. Pouwelse & Garbacki et al. presented Tribler [27], a p2p file sharing system as an extension to Bit-Torrent that uses social context data like friend lists and "taste buddies" in peer selection.

In daily life, people are spending time in classrooms, offices, public transportations with other people around. Technically, it is trivial to create a close proximity ad-hoc network (ON) since any phone now supports at least Bluetooth; some of them even Wi-Fi. In SenseMe, ad-hoc networking is managed automatically but peer selection is up to users with the assistance of the system. Since the purpose of bringing social awareness is having a good selection metric, the question is what kind of information can be presented to user. Fortunately, as mentioned before, SNSs cover lots of information: favorite books, TV shows, movies, music, common friends and events. For example, a user is probably interested in knowing someone who has some similar tastes on books, has a few common friends in Facebook and attending the event on weekend.

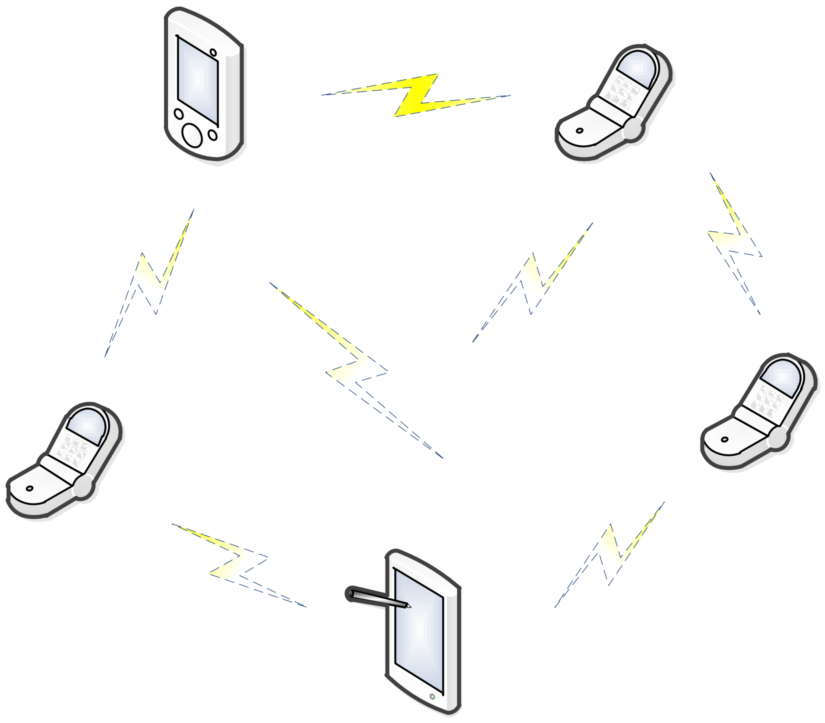


Figure 4 – Peer-to-Peer Applications

Peer-to-Service applications cover user to computer relations in this thesis and contain recommender systems and services that requires user customization (Figure 5). “Recommender systems” recommend items that users might be interested, based on some criteria. Content based systems use user’s previous selected/rated/bought items as criteria while collaborative systems selects items based on other people that have similar tastes [28]. These type of approaches work best on online shopping webpages since users have to register and sites have users’ information. Some services may require user input for customization like age or gender. Both types of systems require user input once with registration or every time. With social context sharing, services can get user data on the fly, which will dramatically improve user experience.

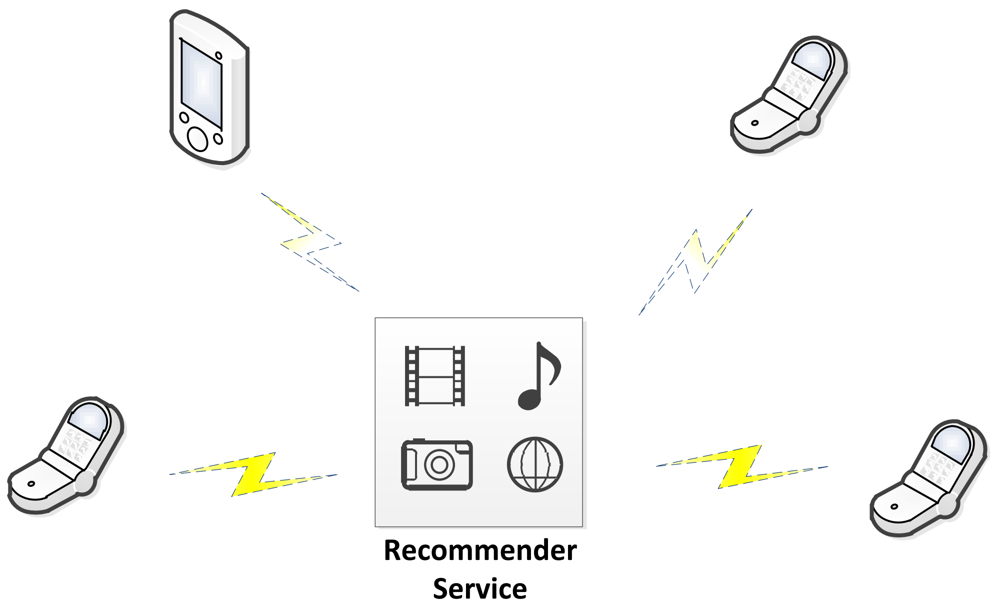


Figure 5 – Peer-to-Service Applications

2.6 Other Works Related With SenseMe

Opportunistic networks and social relations attracted many studies. In this section, some key studies that inspired the thesis will be discussed. Understanding the problems or limitations in previous studies will lead a better system design.

One of the most innovative studies about mobile social networks is Serendipity by Eagle & Pentland [29]. To best of the author’s knowledge, Serendipity is the first proposal to form social relations in face-to-face meetings, using mobiles. Serendipity central server contains user’s profile and matching preferences which user has to register. Peer discovery is handled by Bluetooth; system receives a Bluetooth ID that is sent to the central server to be processed. There are obvious flaws in the system, like the requirement for central server. In addition, profiles are not fetched from any SNS, but users have to register and enter all the information manually. Serendipity relies on device Bluetooth addresses, which is not user-friendly at all.

WhozThat by Beach & Gartrel & Akkala [30] seeks answer to the famous question, “Who’s that guy/girl” in human relations. Unlike Serendipity, WhozThat uses existing social networks instead of populating own context database. In addition, WhozThat focuses on providing an ID sharing framework instead of developing a profile matcher. Social Network ID links are shared by mobile phones using Bluetooth, then devices inquiry SNS with ID by connecting internet for fetching context data. Study demonstrates the framework by implementing a very smart idea: context aware Music Jukebox. The scenario is when WhozThat users are sitting in a bar, context aware jukebox should populate the playlist according to customers' music preferences. One major limitation of WhozThat approach is it requires internet connectivity on devices. SocialAware by Gartrell [31] proposed a recommender framework built on WhozThat and implemented a context-aware music and movie player that dynamically populate playlist according to nearby people.

MobiClique by Pietiläinen & Oliver et al. [32] takes a similar approach, starting system using existing SNS, but without requiring any central server. MobiClique relies on ad-hoc networking and provides a middleware services such as messaging for future applications. However, MobiClique requires smart phone capabilities.

All the previous studies discussed up to now used user-input data, by either populating directly from user or fetching from existing SNS. Another type of social applications like CenceMe (mentioned in *2.2 Context Awareness*) or SocialFusion by Beach & Gartrel et al. [33] populate their own data by processing data sources like sensors. SocialFusion, a very recent study, gathers data from analyzing mobile phone content (call history, contacts etc.), sensor data from fixed sensor networks and finally existing social networks. Again, processing power and input data requires smart phone capabilities for such an application.

Some studies approached social profile sharing from a different perspective, for transferring contact information just like exchanging business cards. SocialCard by Eraslan & Beygo [34] enables users to share contact information just by touching the mobiles, using Near Field Communication technology.

Chapter 3 – SenseMe Mobile Social Context Sharing Application

In the previous part of the study, requirements and specifications of social-aware mobile applications are defined and previous proposals are examined. This chapter introduces SenseMe, a framework that contains:

- Lightweight file format to carry profile data
- Application level protocol to exchange profile data
- Mobile application for discovery of other peers and services

Before advancing further, the motivation behind the thesis should be illustrated in a real life scenario:

A user leaves home to have a coffee with his/her friends. While sitting in the Café, his/her mobile socially interacts with other people's mobiles and discovers new friends, recommendations, trade opportunities. User then decides to visit a bookstore and as he/she approaches the kiosk, kiosk will automatically recognize the user and display customized page.

The presented scenario will be used as use-case for SenseMe. Previous studies all proposed similar solutions but there were issues about central server requirement, smart phone or constant internet connection requirements. For wider market acceptance, more lightweight approach is required; in this aspect the key differences and design goals of SenseMe are:

- Ability to run on regular mobile phones
- Minimum internet connection requirements
- No central server, rely on distributed systems
- Focus on peer-to-peer
- Respect for user privacy

Such requirements bring unique challenges from software development perspective; most of them are solved using existing open source projects. Following chapters contain design decisions and technical details about SenseMe.

3.1 System Overview

System architecture is shaped around two important factors, *2.3.3 Phone Limitations* and *2.4 Privacy and Security Issues*. SenseMe system includes the following components: profile file

that contains social data with application protocol for data exchange *(3.2 File Format and Protocol)*, one web page for profile customization and digital signing *(*
3.3 SenseMe Web*)*, mobile agents *(3.4 SenseMe Mobile)* and third party applications *(3.5 Services)*. For the scenario discussed in the introduction of this chapter, the whole experience can be divided into three phases:

1. **Profile Customization:** The user navigates to SenseMe Web and installs SenseMe Facebook Application, only for first time. SenseMe Web fetches profile information with an easy-to-use GUI, which allows user to customize the data and privacy settings then creates profile file.
2. **Agent Configuration:** The user runs the mobile agent on his/her phone and then downloads the profile file created from the SenseMe Web.
3. **Discovery:** The user runs the mobile agent in the background. Agent discovers other users and services.

3.2 File Format and Protocol

The user's social profile should be carried by a standard file format that allows interoperability between different programs, such as SenseMe Agent and Services. The types of social information are presented in *2.1 Social Networks* and besides some system fields like unique profile id, date of creation and signature are required.

The profile file is designed as a flat structured, text based XML file with UTF-8 encoding. The binary data fields like user's profile picture or signature values are serialized to ASCII string using Base64 Encoding. The reason for flat structure, no nested-tags, is to ease parsing in limited devices, since string operations are expensive. File fields are divided in four logical groups, identity, social, services & trade and validity information. Some fields are required and guaranteed to be non-empty; some files can be empty since user can decide not to share that information. All the fields that are not marked as "Required" are optional in the following fields table.

SenseMe Profile File Fields

Identity Information: *This group contains user's identity data. Since SenseMe is working on PANs, sex and age information are also considered as identity. For example, anyone can be at age 18 or male, but in close proximity, there may be only one person older than 40 so people can visually identify user. Fields in this group should not be sending without user's consent, 2.4 Privacy and Security Issues.*	
Id	**Required.** Unique SenseMe profile ID for user, generated by hashing user's Facebook ID. ID value will be same no matter when the profile is updated.
Fid	User's Facebook ID.
Name	**Required**. User's full name in "Name <Middle Name> Surname" format.
Picture	Base64 encoded stream of user's Facebook profile picture in jpeg format. Picture may be compressed for size concerns in the expense of quality.
birthday	User's birthday in mm/dd/yyyy format.
Sex	User's gender, *male* or *female*.
friendnames	User's friend names. Name <Middle Name> Surname format. Comma separated. No space between commas. This field co-exists with friends field with same ordering: i.e. friendnames[0] contains name of the friend whose ID is friends[0].
Social Information: *This group contains user's social preferences with friend list and considered as anonymous. There is no way to identify users based on favorite books or movies and friend ID's are irreversibly hashed before storing. Hashing of ID's preserves privacy while allowing comparison, if the hashes are same then ID's are same also (Collision of hashing algorithm is ignored, since Facebook ID's are shorter than generated hash).*	

friends	User’s friends’ Facebook IDs. Hashed for preserving privacy. This field co-exists with friendnames field with same ordering: i.e. friends[0] contain ID of the friend whose name is friendnames[0].
activities, TV interests, music movies, books	Lists of user’s favorite items. Comma separated. No space between commas.
Services and Trade: *This group contains data for trading and services. Categories should be cross-compared. Trading category allows user to find a new item or sell/exchange. For example, people usually trade computer/console games. Services category contains user’s interested services or third party application provided services. For greater efficiency in matching, service names can be standardized like “BookDiscounts”, “MovieRecommender” etc.*	
have	List of items that user already have and want to trade. Comma separated. No space between commas.
lookingfor	List of items that user is looking for. Comma separated. No space between commas.
servicesinterested	List of service names that user is interested to be notified. Comma separated. No space between commas.
servicesprovided	This category is only intended for third party applications. List of service names that application offers. Comma separated. No space between commas.
Validation: *This group contains values automatically set by SenseMe website. Agents or Services must use the data to validate profile. Digital signing details like hash algorithms etc. are out of thesis scope. Efficient signing system for feature phones must be studied in order to define a standard.*	
date	Profile creation date. yyyy-mm-ddThh:mm:ss format, CET timezone.
signanon	Digital signature of anonymous data, signed by SenseMe Web private key.

sign	Digital signature of all data, signed by SenseMe Web private key.

Example profile files for users and services are presented in *Attachment 1 – SenseMe Profile File Example for Users* and *Attachment 2 – SenseMe Profile File Example for Services* respectively.

In discovery phase, SenseMe agents share the profile file with other agents (peer-to-peer) and third party applications (peer-to-service). Communication protocol designed to be lightweight and message based. Messages contain a four character message code and zero or many parameters: `[code]<|param1><|param2>`… where **|** character is used as parameter separator.

SenseMe Protocol Messages

helo|<uid>|<date>	First message to send after Bluetooth communication established. Sender agent sends **helo** message with its profile ID and profile creation date. All parameters are required.
reqa	Request for anonymous profile information. Must be responded with a **resa** message. No parameters.
resa|<anon_profile>	Response for **reqa** message contains anonymous parts of the profile. Parameter is required.
reqi	Request for remaining (non-anonymous) profile information. If receiver authorizes sender, must be responded with a **resi** message otherwise must be responded (declined) with a **decl** message. No parameters.
resi|<rem_profile>	Response for **reqi** message contains remaining parts of the profile. Parameter is required.
decl	Response for **reqi** message, states that receiver user does not authorize for identity data communication. No parameters.

mesg|<msg_text>	Message transfer. Sender receives no response. Parameter is required. Message data format to be decided by the application.
done	Request for connection termination. Indicates that sender completed profile sharing and message passing and waiting for the other party to terminate the connection. When both party sends and receives **done** message, Bluetooth connection is closed.

Applications should never respond **reqi** message without user's consent, according to *privacy rule #2.* Other messages such as **helo** and **reqa** are considered safe and can be transferred without permission of user (*privacy rule #4*).

Following sequence diagram (Figure 6) shows two consecutive communications of two agents.

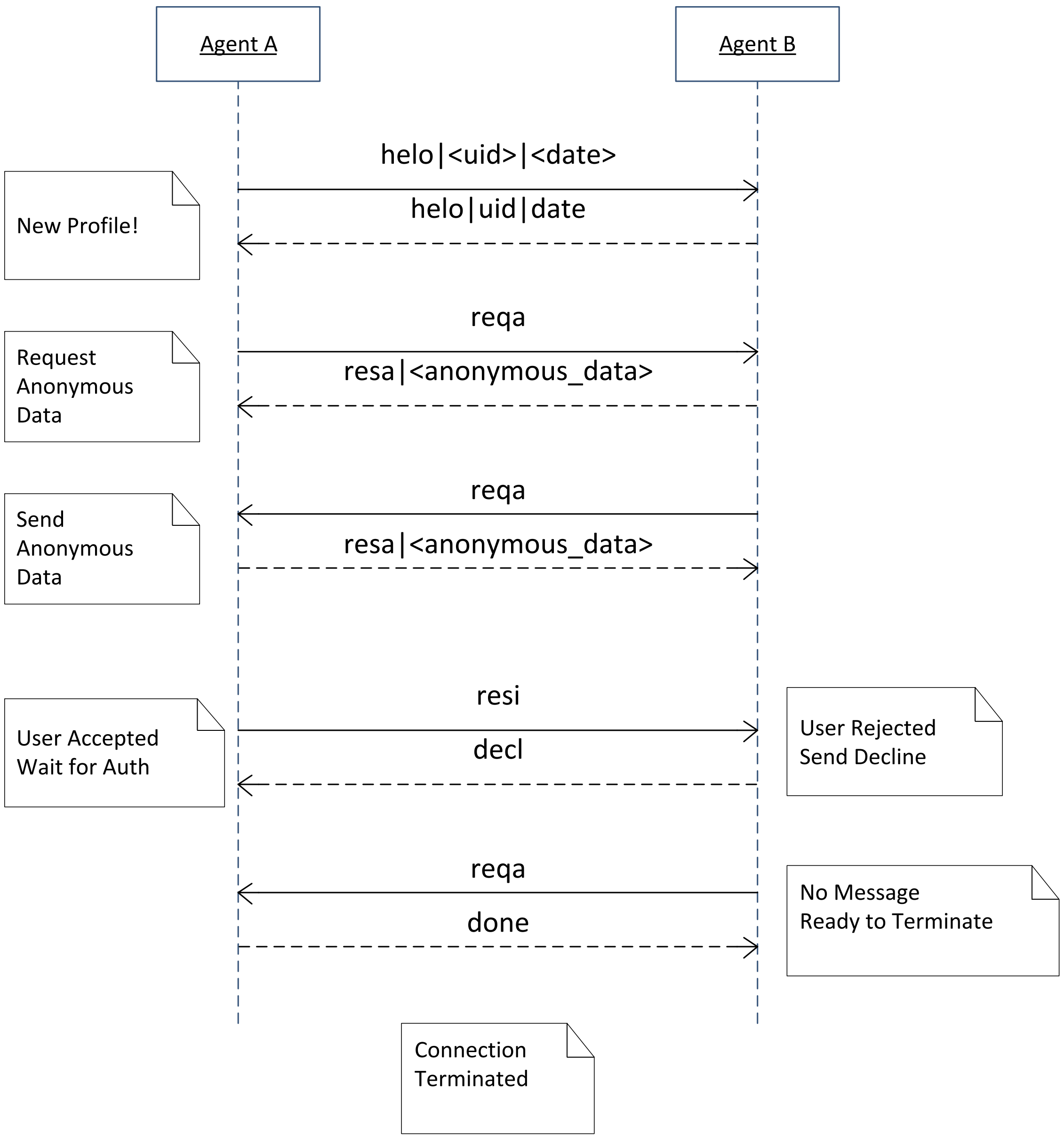


Figure 6 – SenseMe Protocol – Message Sequence Diagram

3.3 SenseMe Web

SenseMe Web is the web page component for profile customization. Profile data is stored and processed on the mobile nodes, but still there are requirements for a web page. Most important three issues why SenseMe requires a website is discussed below.

First, profiles should be signed by a party whose key pair is known to everyone. Agents cannot send their public key along with their profile, since remote party requires knowing the actual public key that belongs to the agent. This problem can be solved like in the internet, where every webpage has a unique id and a trusted certificate authority (SenseMe Web in this case) binds public key to unique id; but this must be investigated further. For this reason, in SenseMe, agents should use the page's public key that is bundled so received profiles can be easily validated.

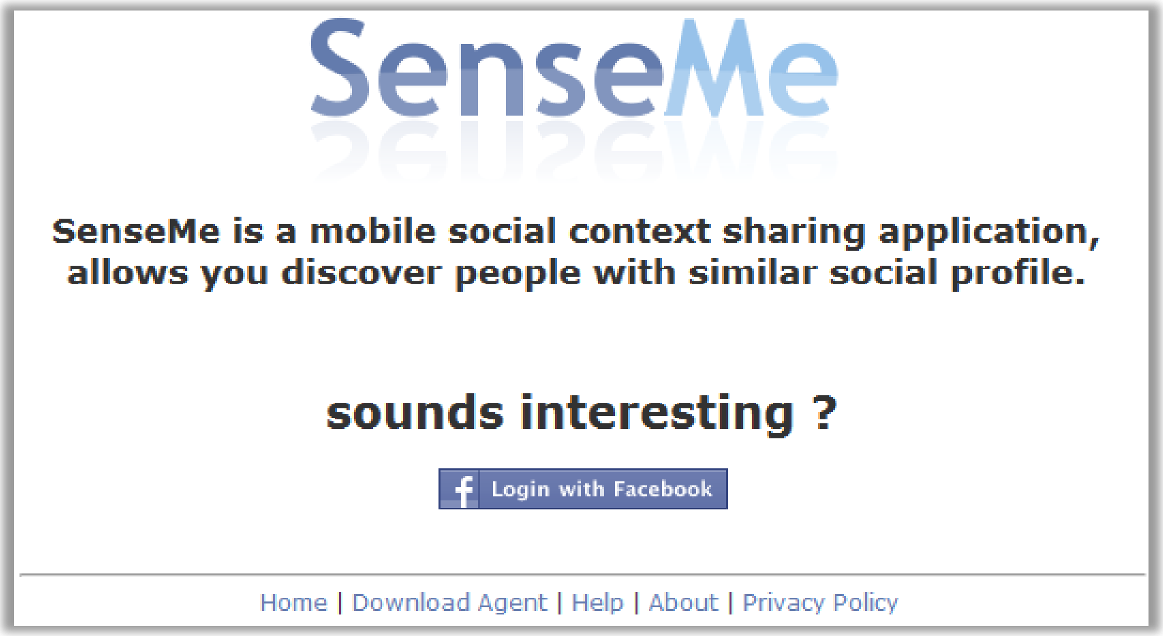
Another issue is the way social networks' integration API's work. In Traditional server - client (3rd party app) authentication model, client stores user's credentials (username and password in this this case) then sends them to the server. For mobile applications, this model introduces some problems:

- For proper user experience, program cannot ask credentials every time. Username and password should be stored in phone storage, which is easily accessible in case the phone is lost
- users cannot be expected to trust third party applications for their Facebook credentials, many user will refuse to use the program

Security of user's credentials brings heavy responsibility for mobile app developers and there is not much can be done. OpenAuth protocol [35] is proposed to solve these problems, where third party applications need access to server resources. In OpenAuth model, client requests access-token on user's behalf and server provides access token when user logs-in the server. For example, client redirects users to the server for logging-in, and when user logs server redirects user to the client with an access token. Facebook Graph API [36] uses OpenAuth mechanism here applications are identified by an **Application ID** and **Application Secret**. As mentioned in *2.4.1 Facebook Privacy Model and Security* if Application Secret is compromised then all the users authorized the application will also be compromised. Without a website, Application Secret must be stored in mobile phones and this leads a catastrophic vulnerability. The only solution is saving the Application Secret on a secured place, like a protected webserver instead of user's hand. Last issue is the user experience. Feature phones have limited screen resolution and input capacity, which will dramatically decrease usability.

For these reasons, a web SenseMe Web is developed using Microsoft .Net Framework for backend, jQuery Framework for client-side and Facebook Open Graph API for integration and open source Json.NET API for JSON parsing.

In **Profile Customization** (defined in *3.1 System*) phase, the website must fetch user's social profile from Facebook and let user customize it. In the welcome page, user is asked to connect with Facebook account by clicking "Login with Facebook" button (Figure 7).


SenseMe

SenseMe is a mobile social context sharing application, allows you discover people with similar social profile.

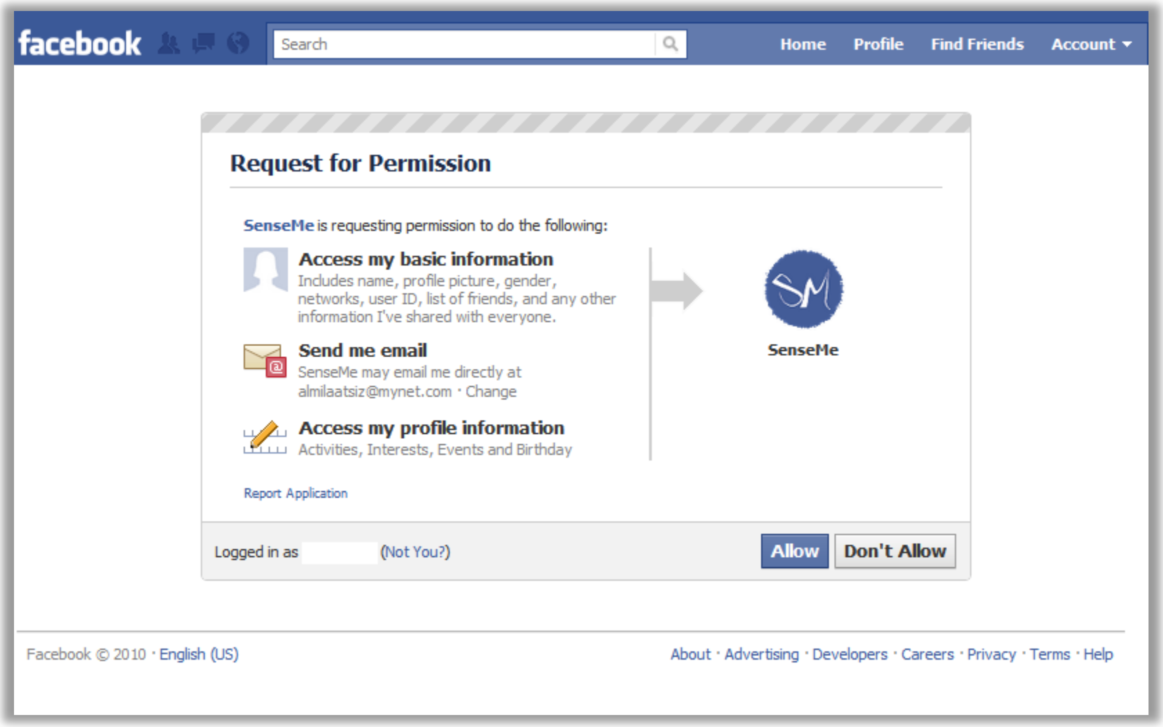
sounds interesting ?

Figure 7 – SenseMe Web – Welcome Page

The button redirects user to Facebook URL:

https://graph.facebook.com/oauth/authorize?client_id=***Application Secret***&redirect_uri=***http://senseme.domain.name/FBCallBack.aspx***&scope=email,user_activities,user_birthday,user_events,user_interests

where user needs to login Facebook and authorize the application, if it is the first use (Figure 8).


facebook

Search

Home Profile Find Friends Account

Request for Permission

SenseMe is requesting permission to do the following:

Access my basic information
Includes name, profile picture, gender, networks, user ID, list of friends, and any other information I've shared with everyone.

Send me email
SenseMe may email me directly at almilaatsiz@mynet.com · Change

Access my profile information
Activities, Interests, Events and Birthday

SenseMe

Report Application

Logged in as (Not You?)

Allow Don't Allow

Facebook © 2010 · English (US)

About · Advertising · Developers · Careers · Privacy · Terms · Help

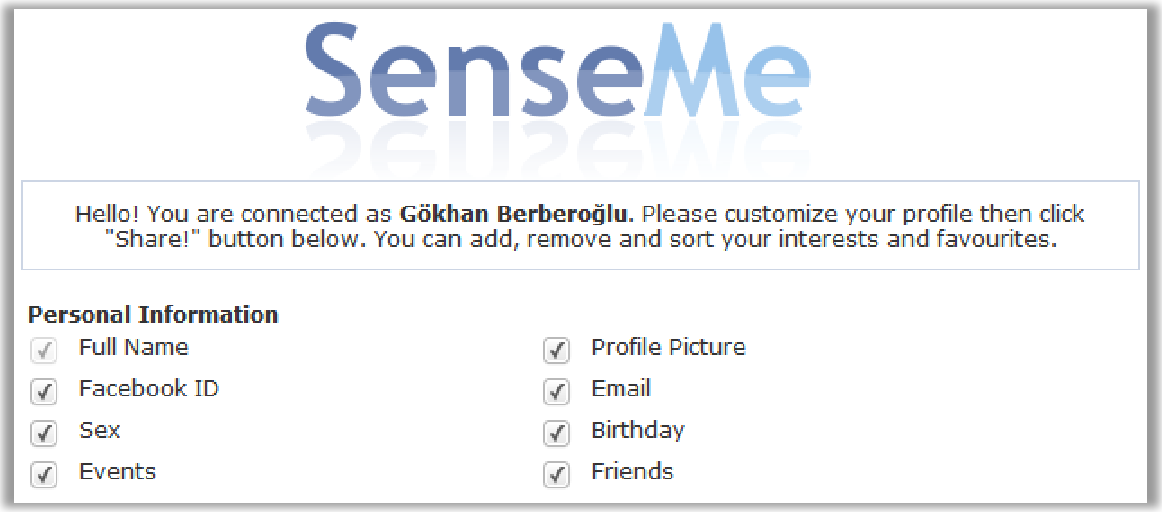
Figure 8 – SenseMe Web – Application Authorization

Once user authorizes the application, Facebook redirects user to the Callback URL in the application domain (http://senseme.domain.name/FBCallBack.aspx in this example) with an **Access Token** parameter. Using this token, without knowing user's Facebook credentials, SenseMe Web can query user's profile.

Callback page fetches the Access Token and silently redirects user to profile customization page, where user can select what to share. Information fetched from Facebook using Graph API calls, which are HTTP GET requests with Access Token and data returns as JSON strings.
For example, to get User's profile object, following URL must be requested:

```
https://graph.facebook.com/userid?access_token=token
```

returned JSON string can be seen in *Attachment 3* – Facebook Graph API User Object JSON. Retrieved string is converted to C# objects using Json.NET library. After fetching complete, data is presented to user for customizing (Figure 9). Identity information like birthday, email or gender cannot be edited but can be excluded from sharing.


Hello! You are connected as **Gökhan Berberoğlu**. Please customize your profile then click "Share!" button below. You can add, remove and sort your interests and favourites.

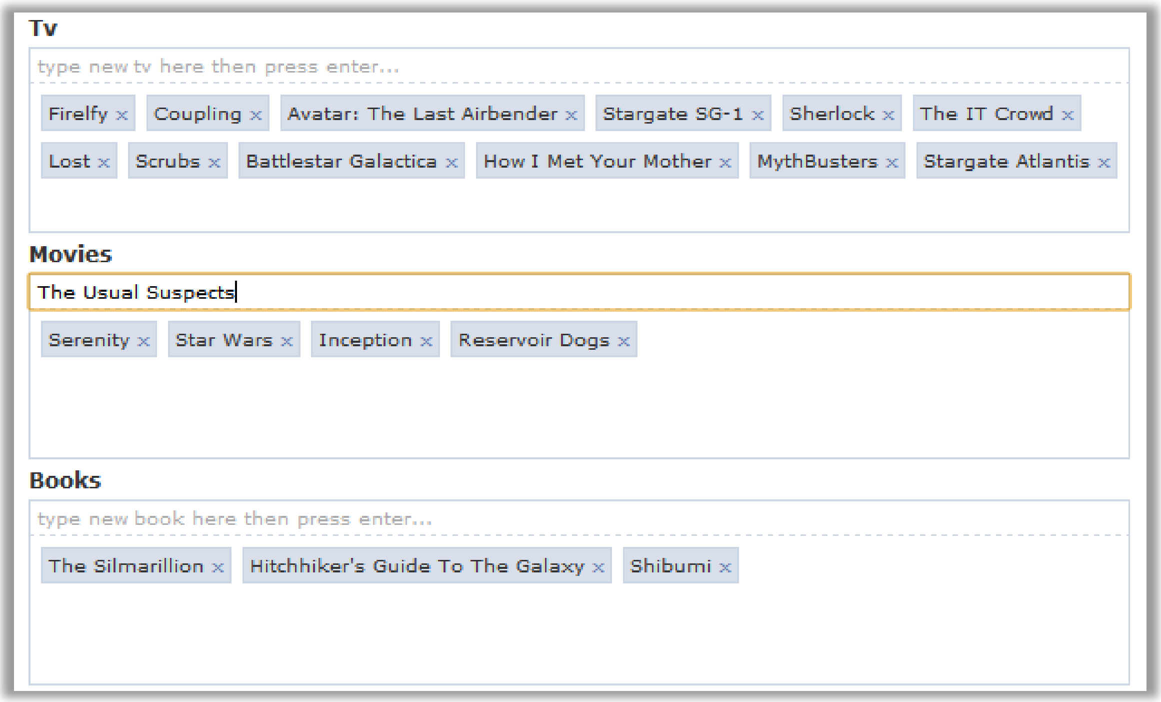
Personal Information

✓ Full Name	✓ Profile Picture
✓ Facebook ID	✓ Email
✓ Sex	✓ Birthday
✓ Events	✓ Friends

Figure 9 – SenseMe Web – Profile Customization A

Users have control on what to share, i.e. Facebook ID or Email in profile file. Only exception is user's name, which is the minimum required information for human perception of "friend". All information fetched from Facebook is presented to user and can be excluded from sharing, according to *2.4 Privacy and* Security Issues.

List items like favorite books, movies or trading are represented as box objects which user can add more, delete or reorder based on preferences (Figure 10). Orders of the items are considered in match algorithm.


Tv

type new tv here then press enter...

Firelfy × Coupling × Avatar: The Last Airbender × Stargate SG-1 × Sherlock × The IT Crowd ×

Lost × Scrubs × Battlestar Galactica × How I Met Your Mother × MythBusters × Stargate Atlantis ×

Movies

The Usual Suspects

Serenity × Star Wars × Inception × Reservoir Dogs ×

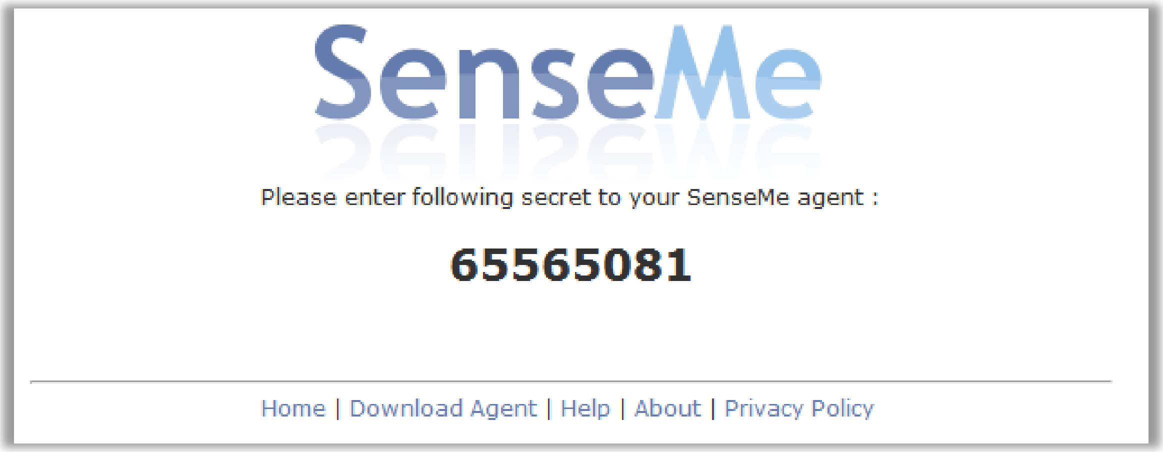
Books

type new book here then press enter...

The Silmarillion × Hitchhiker's Guide To The Galaxy × Shibumi ×

Figure 10 – SenseMe Web – Profile Customization B

After the profile customization is done, user clicks “Share” button in the bottom of the page and user is redirected to the final page. In background, system generates a profile file based on user input and assigns a unique id for it. Unique id is actually name of the profile file, and randomly generated for each profile customization. Unique id is referred as “Shared Secret” in SenseMe, since it should be known only by the profile creator (Figure 11).


SenseMe

Please enter following secret to your SenseMe agent :

65565081

Home | Download Agent | Help | About | Privacy Policy

Figure 11 – SenseMe Web – Shared Secret

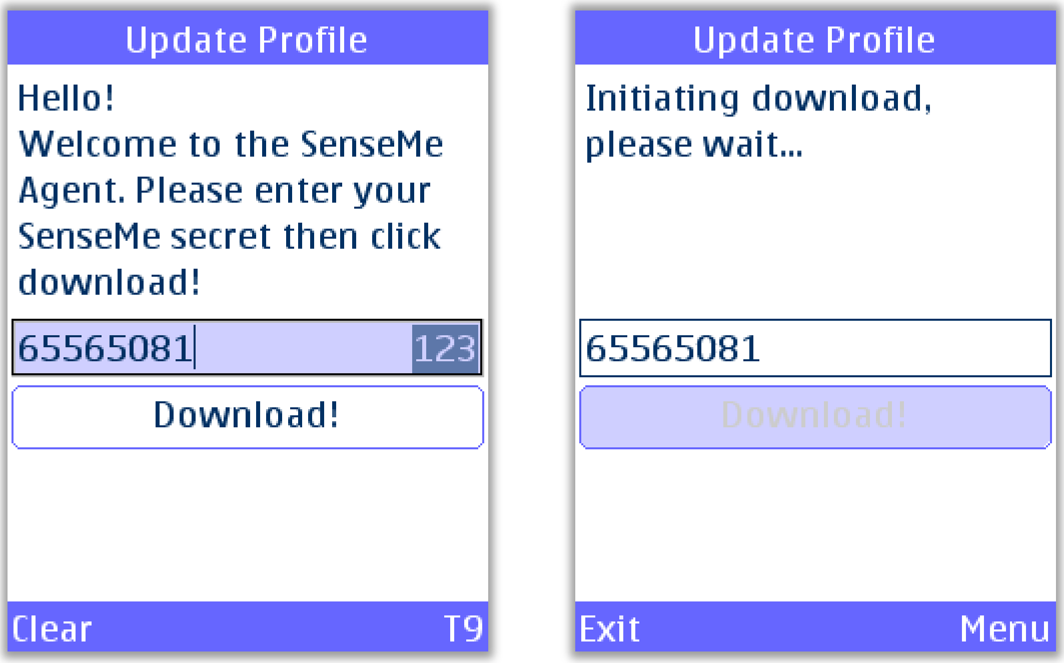
Secret is shared between the SenseMe Web and SenseMe Mobile, allows mobile agent to download correct profile file. The website does not run any algorithm or store any data other than profile information for a short period. Once mobile agent downloads the profile, stored data can be deleted from the website.

3.4 SenseMe Mobile

SenseMe Mobile is the phone application part of SenseMe. Design goal requires a distributed system and minimum internet connectivity, which means all the data must be stored and processed on the phone itself. Required agent features are downloading profile file, finding other peers and services, maintaining a friend list for matched peers and updating user profile.

JavaME platform is selected for development and Bluetooth protocol is selected for communication, because of the wide deployment. JavaME platform lacks of many features as compared to modern smartphones, such as storage, xml parsing, rich GUI ability. Despite all the limitations, development of the application that meets all the requirements is possible thanks to the open source frameworks. User Interfaces are developed using Light Weight UI Toolkit (LWUIT) from Oracle, fast xml parsing uses code from XParseJ by Michael Classen and object persistence uses Floggy Persistence Framework.

According to the *3.1 System ;* in **Agent Configuration** phase, agent must get the data from SenseMe Web. Collaboration between agent and website is done by **Shared Secret**. Agent can be configured easily in a single step. Upon first run, Agent displays a welcome screen (**Update Profile** form) that requires user to enter shared secret generated by SenseMe Web then downloads profile file (Figure 12).


Update Profile

Hello!
Welcome to the SenseMe Agent. Please enter your SenseMe secret then click download!

65565081 | 123

Download!

Clear | T9

Update Profile

Initiating download, please wait...

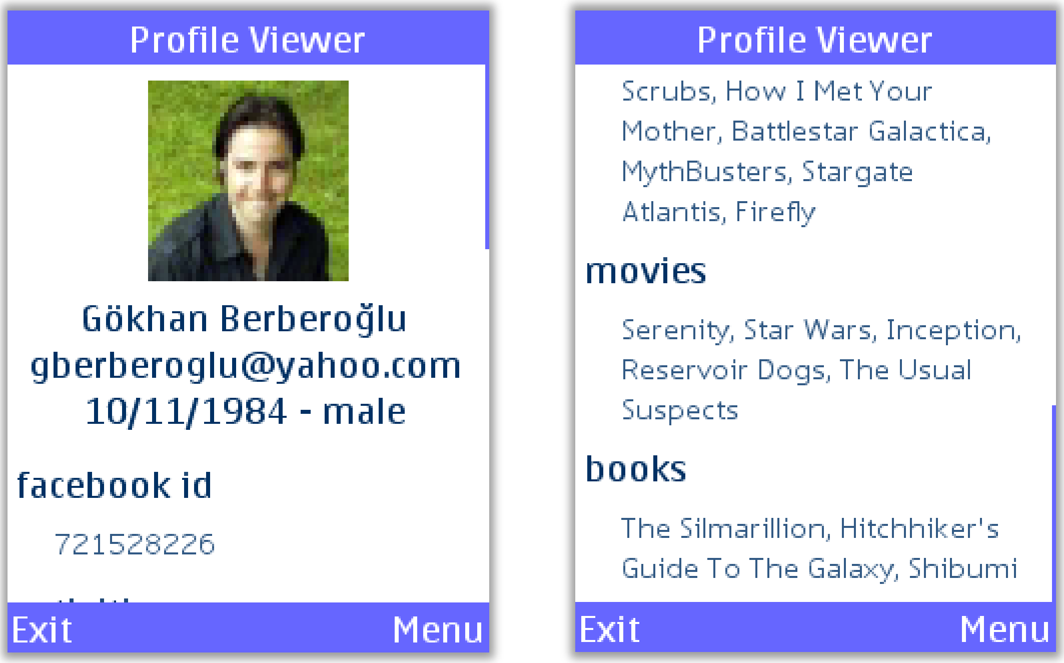
65565081

Download!

Exit | Menu

Figure 12 – SenseMe Agent – Profile Downloading

Users are able to see their profile using **Profile Viewer** form (Figure 13). Profile cannot be edited directly from the phone because the signature is handled SenseMe Web and most of the data must be fetched from Facebook. A new profile must be created and downloaded from the SenseMe web in order to update profile.


Profile Viewer

Gökhan Berberoğlu
gberberoglu@yahoo.com
10/11/1984 - male

facebook id

721528226

Exit | Menu

Profile Viewer

Scrubs, How I Met Your Mother, Battlestar Galactica, MythBusters, Stargate Atlantis, Firefly

movies

Serenity, Star Wars, Inception, Reservoir Dogs, The Usual Suspects

books

The Silmarillion, Hitchhiker's Guide To The Galaxy, Shibumi

Exit | Menu

Figure 13 – SenseMe Agent – Profile Viewer

After downloading the file, agent does not require any internet connection until next profile update and becomes ready for **Discovery** phase. As a limitation of Bluetooth, devices cannot accept incoming connections while inquiring other devices so devices should do a listening / inquiring cycle. Users can decide the Bluetooth behavior, by setting “Discovery Mode” option in **Settings** form, possible values are: “Listen Only”, “Search Only” and “Listen & Search”. For battery concerns, device inquiry is performed every 4 minutes. After inquiry, background task sleeps a random period between 0 and 10 seconds to avoid synchronization (Figure 14). Duration of inquiry depends on JavaMe implementation of the mobile phone manufacturer.

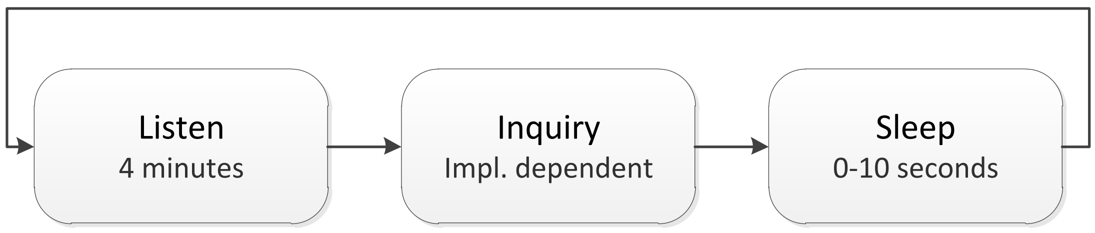
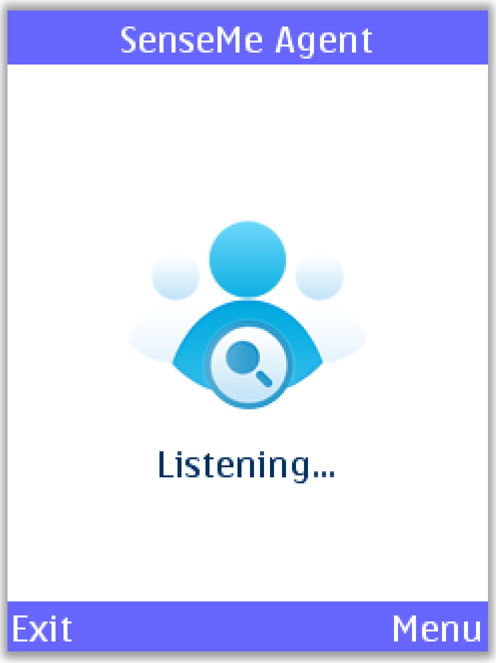


Figure 14 – SenseMe Agent – Run Cycle

For data communication, RFCOMM is used. Listener phone runs a threaded server, so multiple simultaneous connections are possible. Inquiring phone first detects other SenseMe Agents, then connects one-by-one synchronously (Figure 15).

User interaction when connecting to other devices depends on the phone manufacturer, i.e. Nokia phones ask user permission as “Do you want to connect to <device_name>” but Sony Ericsson phones allow silent connection, once required access is given to the application. Best user experience will be requiring only one user interaction per connection, where users authorize each other depending on the profile.



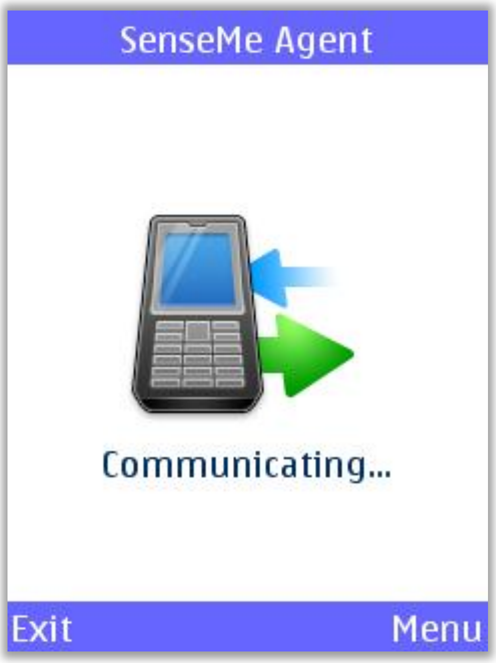


Figure 15 – SenseMe Agent – Device Communication

Devices can run multiple Bluetooth applications simultaneously where each application (also called service in Bluetooth terminology) is identified by an UUID number.

SenseMe Bluetooth UUID = 0118e610-d9f8-11df-937b-0800200c9a66

After connection, agents exchange **helo** messages according to the *3.2 File Format and Protocol*. If remote profile is new or existing but updated, anonymous parts are requested by a `reqa` message. After receiving anonymous parts, agent finds common items and calculates social match percentage. Match percentage is calculated only with social profile categories: activities, TV, interests, music, movies, books, friends (unordered) and events (unordered). Orders of matched items are taken into consideration, since users can reorder items while creating the profile.

Without ordering, match percentage calculation would be simple: number of all items in all categories divided by number of matched items. Since item order is only valid in container category, calculations must be done per category. Calculation algorithm is as follows:

1. Define global counters, total points (*totalPoints*) and match points (*matchPoints*) and set to 0.
2. Define item weight (*itemValue*) and index bonus weight (*itemIndexBonusValue*) set to 1 and 0.3 respectively, which means ordering has plus 30 percent value.
3. For each category, calculate index bonus (*indexBonus*) based on index bonus weight and category item count. Increment total points as if all items matched.
    a. For each item, increment match points by item value plus calculated index bonus.
4. **4.** Calculate percentage by divide match points by total points.

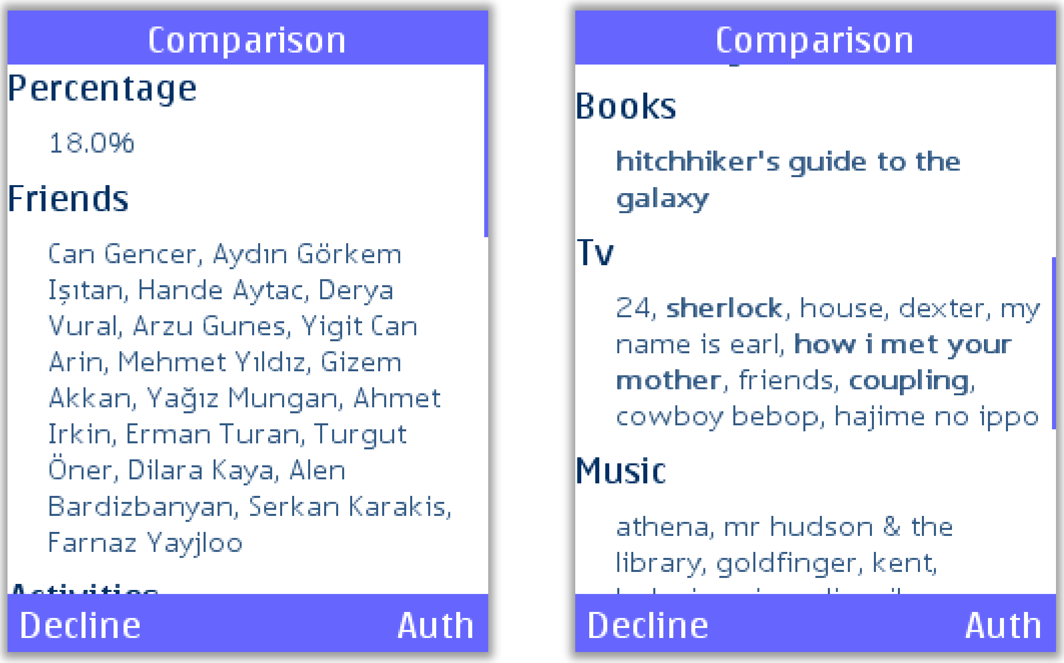
Profile Comparison Algorithm

```
totalPoints := 0;
matchPoints := 0;
itemValue := 1;
itemIndexBonusValue := 0.3;

Compare(LocalProfile, RemoteProfile)

    CommonBooks := FindCommons(Profile1.Books, Profile2.Books)
    CommonMusic := FindCommons(Profile1.Music, Profile2.Music)
    …

    percentage := (totalPoints / matchPoints) * 100

FindCommons(LocItems, RemItems)

    if LocItems = ∅  then     return ∅

    Commons := ∅

    totalPoints += (LocItems.Size * itemValue)
    totalPoints += (LocItems.Size * itemIndexBonusValue)

    indexBonus := (itemIndexBonusValue * 2) / (LocItems.Size + 1)

    forall item ∈ LocItems do
        if RemItems contains item
            matchPoints += itemValue
            matchPoints += (LocItems.Size - LocItems.IndexOf(item)) * indexBonus
            Commons := Commons ∪ item

    return Commons
```

Following table presents different results based on itemIndexBonusValue. It is clear that value 1 makes items either too valuable or too invaluable. By experimenting different bonus values, 0.3 is selected since results are more balanced for different sizes of categories.

Local items	Remote items	Percentage (value = 0)	Percentage (value = 0.3)	Percentage (Value = 1)
1,2,3,4	1	25	28.5	32.5
1,2,3,4	4	25	21.5	17.5
1,2,3…18,19,20	1	5	6.05	7.3
1,2,3…18,19,20	20	5	3.95	2.7

After profile comparison, agents compare and display remote profile with a percentage and ask user if he/she authorizes other user (Figure 16). Note that names under "Friends" category refer to common Facebook friends. If users are attending same events, they are also listed under "Events" category. Finally, if their "lookingfor" and "have" categories have any match in cross-comparison, items are listed under "trade" section.



Comparison

Percentage

18.0%

Friends

Can Gencer, Aydın Görkem Işıtan, Hande Aytac, Derya Vural, Arzu Gunes, Yigit Can Arin, Mehmet Yıldız, Gizem Akkan, Yağız Mungan, Ahmet Irkin, Erman Turan, Turgut Öner, Dilara Kaya, Alen Bardizbanyan, Serkan Karakis, Farnaz Yayjloo

Decline Auth

Comparison

Books

hitchhiker's guide to the galaxy

Tv

24, **sherlock**, house, dexter, my name is earl, **how i met your mother**, friends, **coupling**, cowboy bebop, hajime no ippo

Music

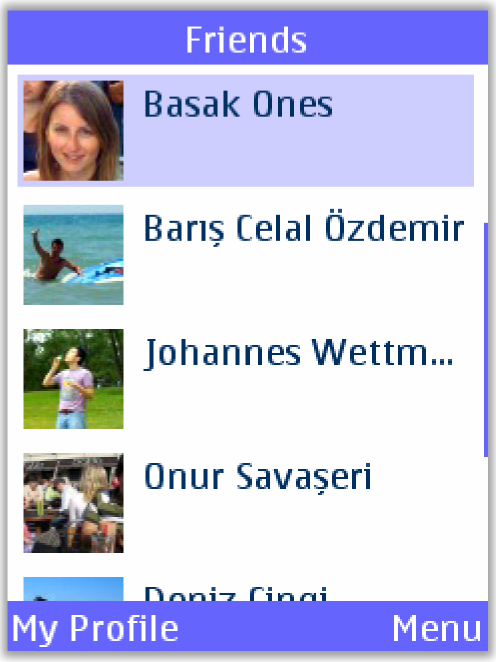
athena, mr hudson & the library, goldfinger, kent,

Decline Auth

Figure 16 – SenseMe Agent – Profile Comparison

If both users authorize each other, remaining identity data is exchanged and remote user is added to local friend list (Figure 17). In contrast to the Facebook friends, local friends are only stored on the device itself and not synchronized to any SNS. Local friend list is sorted by a priority value, initially equals to match percentage. Users are free to reorder friends or delete them from the list without effecting existing friendship relations in the SNS. All the profiles that are stored on the phone memory are available to the Java Runtime, between 256K-1024K bytes in modern phones. Size of a profile file mostly depends on number of friends, average 5KB with 70 friend and 15KB with 500 friends including the picture. Applications need to compress profile data before saving and old profiles with lower priority can be deleted to provide free precious storage space.

Agents are also capable of storing messages for a long time. When user deletes someone from his/her friend list, phone will store a notification message for deleted user and on next communication (if any) the message will be delivered. The message delivery functionality can be further extended using routing based on friendship relations. Assume that Agent A has a message for Agent B, since the network is opportunistic there is no guarantee for further communication opportunity, it may be next moment, may take days or never occur. Therefore, to increase the chance of delivery and minimize delivery time, Agent A can use other agents who have friendship relation with Agent B, i.e. Agent C and Agent D. Users have globally unique ID's, so it is easy to detect if an agent has relationship with given user. For achieving this functionality, messages must be assigned unique IDs; also, there must be a validation mechanism of message source and integrity.



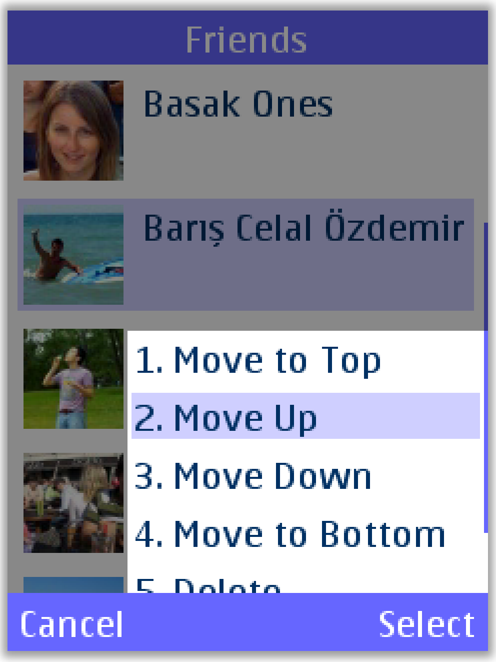


Figure 17 – SenseMe Agent – Friend List

3.5 Services

Services are third party systems that require user’s social profile for customization or recommendation purposes. They can easily integrate with SenseMe agent thanks to the Bluetooth and simple xml based protocol. Services are supposed to actively search for SenseMe GUID and query agents by sending their own profiles like any other agent.

Service profiles do not contain any social information so a social matching is not possible with agents; instead, **servicesinterested** and **servicesprovided** categories will be cross-compared. Depending on the settings, agent will decline service’s request, share anonymous profile or share full profile.

Chapter 4 – Results and Discussion

User experiments are conducted in order to measure prototype application's social worth and usability. This section presents an experiment for the peer-to-peer aspect of SenseMe and discussion of the results.

Ten users, mostly university students, were selected and given introduction to the SenseMe system. Users, gathered in crowded cafeteria of Chalmers University of Technology, were asked to create a profile through SenseMe Web and download it to the phones. As in the scenario mentioned before a short time after users run the application, agents started to discover other agents and displayed profile comparisons. Since some of the test subjects already knew each other, "common Facebook friends" section is removed from the comparison screen to improve the anonymity of the remote party.

Although users were motivated to enter more, average number of social items per profile was 25.2 with a standard deviation of 14.05. Following table presents profile match percentages and whether the user authorized remote user after comparison. Rows represent user himself/herself and columns represent remote user. If a user authorized remote party, percentage is displayed with an underline.

	UserA	UserB	UserC	UserD	UserE	UserF	UserG	UserH	UserI	UserJ
UserA		0	0	9	0	9	7	0	0	19
UserB	0		0	3	3	0	0	5	2	0
UserC	0	0		9	0	0	0	0	0	0
UserD	4	8	4		0	7	3	7	4	18
UserE	0	6	0	0		0	0	6	0	0
UserF	5	0	0	8	0		0	5	0	11
UserG	4	0	0	6	0	0		0	0	4
UserH	0	6	0	7	4	5	0		0	0
UserI	0	14	0	14	0	0	0	0		0
UserJ	6	0	0	11	0	8	4	0	0	

It is also important to see how the relations are formed. The diagram below illustrates relations between the users after the experiment was completed. Solid line represents a formed friendship relation; a directed dashed line represents a one-way authorization from source to destination.

During the experiment, users were able to socially interact with each other despite the limited input and processing capabilities of feature phones. Users expressed that they felt no pressure for declining remote party since only anonymous parts are sent for comparison. From the usability perspective, most common complaint was connection confirmations on Nokia phones which interrupts users every time when a connection to be made.

Users with a lower number social items like C and I were not a popular candidate for friendship relation. Despite the low match percentages, 8 new friendship relations can be formed, which means 18% chance (16 authorized comparisons out of 90) of finding new peers (Figure 18).

In real life, meeting 9 other SenseMe users in same cafe is unrealistic but still the results are very promising for the prototype implementation. It is clear that new friendship relations will be formed even if the match percentage maybe low, since users are evaluating remote profile with the whole social items, not just common properties. For the long-term usage, users will form a new peer-to-peer overlay network according to their social preferences.

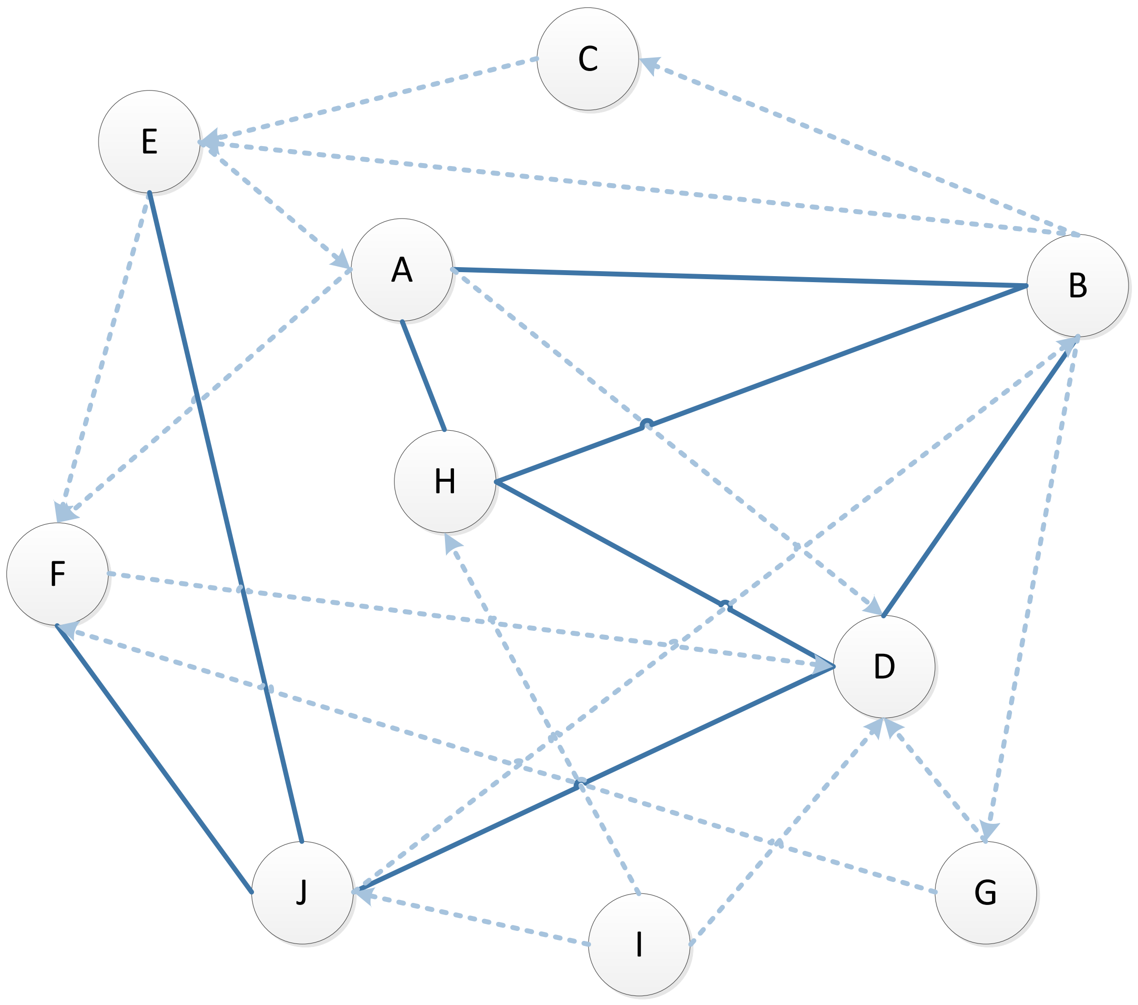


Figure 18 – User Relationship Diagram

Chapter 5 – Conclusion

In this thesis, a mobile social application is presented that adds social layer to opportunistic networks and converts regular mobile phones to social agents. Important properties of the system are the distributed design, ability to work offline and usage of the existing SNS profiles.

Using SenseMe, users can discover people with similar profile and share social profile with context-aware applications in a privacy respected, safe environment. As the time goes by, each agent will have list of opportunistically met people in addition to Facebook friends, which is also a social overlay network for future applications. Storing such rapidly expanding information on the phone will improve many solutions discussed in *2.3.1 Opportunistic Networks*, such as new generation trust systems. For the services like recommender systems, SenseMe provides user's social data just by querying using a simple protocol via Bluetooth, hopefully will inspire new kinds of social-aware applications.

The prototype application is implemented using JavaMe and peer-to-peer aspects were validated in a real life experiment, which shows social-awareness does not requires powerful hardware with fully featured operating systems. By using such lightweight and distributed approach, social-awareness can be implemented on embedded systems or sensor networks also. During the design and implementation of the system, different parts of the problem i.e. privacy and technical are discussed. Many inspiring ideas/features that cannot be executed due to time limit can be found in *Chapter 6* – Future Work.

Chapter 6 – Future Work

The prototype system does not include **profile-signing feature**. As mentioned in
3.3 SenseMe Web signing can be done in two ways, by SenseMe Web site with pre-installed SenseMe Web certificate to agents or each agent have its own certificate issued by SenseMe Web. Latter requires more research, i.e. to build a trust system.

Also in the prototype, **matching** is done by basic string matching. Since social items are language dependent, i.e. movie names are different in every country; profiles must be processed to achieve proper matching. This task requires strong algorithms and many online resources (i.e. movie database) this work must be handled by SenseMe Web, before profiles are downloaded by agents. Matching efficiency can be improved further by allowing peers to choose more sophisticated individual preference metrics other than ordering of items. In that regard, the distributed matching algorithm presented by Georgiadis & Papatriantafilou [37], allows peers with their own preference list to form an overlay network without exposing the metric itself, can be adopted to SenseMe.

Since agents have no guarantee to meet each other again, inter agent communication i.e. notification messages are highly undependable. This feature can be improved by **message queuing in SenseMe Web**; every agent can connect SenseMe Web when internet is available, and send the cached messages and receive messages from other agents. Still, there can be no delivery time guarantee for the messages but reliability will be much higher.

Another interesting feature to implement is **distributed querying/searching** among agents. Since each agent has its own copy of friend profiles, agents can process received search queries on their own profile database and forward it to other connected or future agents. By using created overlay network, query distributing will be greatly promising.

Bibliography

1. **Facebook.** *Facebook Statistics.* [Online] [Cited: 09 05, 2010.] http://www.facebook.com/press/info.php?statistics.

2. *LoKast.* [Online] NearVerse. [Cited: 09 06, 2010.] http://www.nearverse.com/lokast.

3. *Social Network Sites: Definition, History, and Scholarship.* **Boyd, Danah M. and Ellison, Nicole B.** 1, s.l. : Journal of Computer-Mediated Communication, 2008, Vol. 13.

4. **Karinthy, Frigyes.** *Everything is different, Chains.* Hungary : s.n., 1929.

5. Study revives six degrees theory. *News.* [Online] BBC, August 3, 2008. [Cited: 09 08, 2010.] http://news.bbc.co.uk/2/hi/technology/7539329.stm.

6. *Identity Management : Multiple Presentations of Self in Facebook.* **Dimicco, Joan Morris and Millen, David R.** s.l. : ACM, 2007. Supporting Group Work.

7. *All My People Right Here , Right Now : Management of Group Co-Presence.* **Lampinen, Airi, Tamminen, Sakari and Oulasvirta, Antti.** s.l. : ACM, 2009. Proceedings of GROUP '09.

8. *Student socialization in the age of facebook.* **Barkhuus, Louise and Tashiro, Juliana.** s.l. : ACM, 2010. Human Factors in Computing Systems.

9. *Context-Aware Computing Applications.* **Schilit, Bill N., Adams, Norman and Want, Roy.** s.l. : IEEE, 1994. Workshop on Mobile Computing Systems and Applications.

10. *Understanding and using context.* **Dey, Anind K.** s.l. : ACM, 2001. Personal and Ubiquitous Computing.

11. *CenceMe – injecting sensing presence into social networking applications.* **Miluzzo, Emiliano, et al.** s.l. : Springer Verlag, 2007.

12. *Disseminating active map information to mobile hosts.* **Schilit, Bill N. and Theimer, Marvin M.** 5, s.l. : IEEE, 1994, Vol. 8.

13. *Opportunistic networking: data forwarding in disconnected mobile ad hoc networks.* **Pelusi, Luciana, Passarella, Andrea and Conti, Marco.** 11, 2006, Communications Magazine, IEEE, Vol. 44.

14. **Group, Delay Tolerant Networking Research.** [Online] Internet Research Task Force. http://www.dtnrg.org.

15. Facebook Places. [Online] Facebook. [Cited: 9 18, 2010.] http://www.facebook.com/places/.

16. *Bubble rap: social-based forwarding in delay tolerant networks.* **Hui, Pan, Crowcroft, Jon and Yoneki, Eiko.** s.l. : ACM, 2008.

17. **Conti, Marco and Kumar, Mohan.** Opportunities in opportunistic computing. *Computer.* 2010, Vol. 43, 1.

18. *Bluetooth and Wi-Fi wireless protocols: a survey and a comparison.* **Ferro, Erina and Potorti, Francesco.** 1, s.l. : IEEE Wireless Communications Magazine, 2005, Vol. 8.

19. *Social Networks Security.* **Nagy, Jan and Pecho, Peter.** s.l. : IEEE, 2009. Third International Conference on Emerging Security Information, Systems and Technologies.

20. *Privacy Concerns and Information Disclosure: An illusion of control hypothesis.* **Brandimarte, Laura, et al.** s.l. : Carnegie Mellon University, 2009.

21. *Information Revelation and Privacy in Online Social Networks.* **Gross, Ralph and Acquisti, Alessandro.** s.l. : ACM, 2005. Workshop on Privacy in the Electronic Society.

22. *Developing privacy guidelines for social location disclosure applications and services.* **Iachello, Giovanni, et al.** s.l. : ACM. Symposium on Usable privacy and security.

23. *All your contacts are belong to us.* **Bilge, Leyla, et al.** s.l. : ACM, 2009. 18th international conference on WWW.

24. **Boyd, Danah and Hargittai, Eszter.** Facebook privacy settings: Who cares? *First Monday.* 2010, Vol. 15, 8.

25. Facebook Devleoper Policies. [Online] Facebook. [Cited: 09 19, 2010.] http://developers.facebook.com/policy/.

26. Visa Makes NFC Mobile POS Payments Commercially Available for the First time. [Online] Visa. [Cited: 09 14, 2010.] http://corporate.visa.com/media-center/press-releases/press921.jsp.

27. *Tribler: A social-based peer-to-peer system.* **Pouwelse, J, Garbacki, P and Wang, J et al.** 2008.

28. *Fab: content-based, collaborative recommendation.* **Balabanović, M and Shoham, Yoav.** 3, s.l. : ACM, 1997, Vol. 40.

29. *Social Serendipity: Mobilizing Social Software.* **Eagle, Nathan and Pentland, Alex.** 2, s.l. : IEEE, 2005, Vol. 4.

30. *Whozthat? evolving an ecosystem for context-aware mobile social networks.* **Beach, Aaron, et al.** 4, s.l. : IEEE, 2007, Vol. 22.

31. *Socialaware: Context-aware multimedia presentation via mobile social networks.* **Gartrell, Charles.** s.l. : University of Colorado, 2008.

32. *MobiClique: Middleware for Mobile Social Networking.* Pietilainen, Anna-Kaisa; Varghese, George; Oliver, Earl; Lebrun, Jason; Diot, Cristophe : ACM, 2009.

33. *Fusing mobile, sensor, and social data to fully enable context-aware computing.* **Beach, Aaron, et al.** s.l. : ACM, 2010. Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications.

34. **Eraslan, Cihan and Beygo, Kerem.** *Enhancing Security and Usability Features of NFC.* s.l. : LAMBERT Academic, 2009.

35. The OAuth 2.0 Protocol. [Online] [Cited: 10 13, 2010.] http://tools.ietf.org/html/draft-ietf-oauth-v2-10.

36. Graph Api. [Online] Facebook. [Cited: 10 13, 2010.] http://developers.facebook.com/docs/api.

37. *Overlays with preferences: Approximation algorithms for matching with preference lists.* **Giorgos Georgiadis, Marina Papatriantafilou.** s.l. : 2010 IEEE InternationalSymposium on Parallel & Distributed Processing (IPDPS), 2010. pp. 1-10.

Attachments

Attachment 1 – SenseMe Profile File Example for Users

```
<?xml version="1.0"?>

<sensemeprofile v="1.0">

<!-- Identity Information -->
<id>7A7750861258361E</id>
<fid>721528226</fid>
<name>Gökhan Berberoğlu</name>
<picture>/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAoHBwgHBgoICAgLCgoLDhgQDg0ND
h0VFhEYIx8lJCIfIiEmKzcv7Ozv...</picture>
<birthday>10/11/1984</birthday>
<sex>male</sex>
<friendnames>Friend 1, Friend 2...</friendnames>

<!-- Social Information -->
<friends>DC5677A25ACABEE1,A29EF809942F2253...</friends>
<activities>Activity 1,Activity 2...</activities>
<interests>Interest 1,Interest 2...</interests>
<music>Music 1,Music 2...</music>
<tv>Tv Show 1,Tv Show 2...</tv>
<movies>Movie 1,Movie 2...</movies>
<books>Book 1,Book 2</books>


<!-- Services and Trade -->
<have>Item 1,Item 2...</have>
<lookingfor> Item 1,Item 2...</lookingfor>
<servicesinterested>Service 1,Service 2...</servicesinterested >

<!-- Validation -->
<date>2010-09-06T01:56:19</date>
<signanon>nihUFQg4mDhLgecvhIcKb9Gz8VRTOlw+adiZOBBXgK4JodEe5aFfCqm8WcRI
T8GLLXSk8PsUP4//SsKqUBQkpot...</signanon>
<sign>cAqQAhtz2v9kCWdoUDnAOtFZkd/CnsZ1sge0ndha40wWDV+nOWyJxkYgicvB8POY
tSmldLLepPGMz+J7/UwsBQkpot=...</sign>
</sensemeprofile>
```

Attachment 2 – SenseMe Profile File Example for Services

```
<?xml version="1.0"?>

<sensemeprofile v="1.0">

<!-- Identity Information -->
<id>DAA75EA6125B3614</id>
<name>BookStore Recommender</name>

<!-- Services and Trade -->
<servicesoffered>Service 1,Service 2...</ servicesoffered >

<!-- Validation -->
<date>2010-08-05T13:21:10</date>
<signanon>ni4//SsKqUBQkpotodEe5aFfChUFQg4mDhLgecvhIcKb9Gz8VRTOlw+adiZO
BBXgK4Jqm8WcRIT8GLLXSk8PsUP...</signanon>
<sign>ihUFQg4mDhLgecvhIcd/CnsZ1sge0ndha4LgecvhIcKbKb9GzZk9Gz0wWDV+nOWy
JxkYgicvYtSmldLLepPGMz+J7/B8POUwsBQkpot=...</sign>
</sensemeprofile>
```

Attachment 3 – Facebook Graph API User Object JSON

```
{
  "id": "721528226",
  "name": "Gökhan Berberoğlu",
  "first_name": "Gökhan",
  "last_name": "Berberoğlu",
  "link": "http://www.facebook.com/gokhan.berberoglu",
  "birthday": "10/11/1984",
  "hometown":
  {
    "id": "106478736056198",
    "name": "Ankara, Turkey"
  },
  "location":
  {
    "id": "110448935649941",
    "name": "Göteborg, Sweden"
  },
  "email": "gberberoglu@yahoo.com",
  "website": "http://berberoglu.gen.tr",
  "timezone": 2,
  "locale": "en_US",
  "verified": true,
  "updated_time": "2010-10-06T22:14:24+0000"
}
```

Attachment 4 – SenseMe Agent Source Code Excerpts

Following are excerpts of SenseMe Agent source codes. Project requires CLDC-1.1 and MIDP-2.0 profile.

Source Code Structure:

```
Senseme
     Program.java
     bluetooth
          Client.java
          Event.java
          Handler.java
          Server.java
          ServiceFinder.java
     gui
          BaseForm.java
          DiaProfileMatchViewer.java
          FrmAgent.java
          FrmFriendsViewer.java
          FrmProfileDownloader.java
          FrmProfileViewer.java
          FrmSettings.java
          YesNoDialog.java
     logic
          BackgroundTask.java
          CacheManager.java
          ConfigurationManager.java
          MessageQueueManager.java
          ProfileManager.java
          ProfileMatcher.java
          Protocol.java
          StateEvent.java
     model
          Cache.java
          Configuration.java
          MessageQueue.java
          Profile.java
     util
          Base64Coder.java
          Comparable.java
          Helperrs.java
          HttpDownloader.java
          QuickSorter.java
```

bluetooth/Server.java

```java
public class Server extends Thread
{
    public static final String UUID_STRING = "0118e610d9f811df937b0800200c9a66";
    public static final String SERVICE_NAME = "echoserver";
    private boolean isRunning = false;
    private StreamConnectionNotifier server;
    private Vector handlers;

    private Event event;

    public Server(Event event)
    {
        this.event = event;

        Log.p("BluetoothServer()", Log.DEBUG);
        handlers = new Vector();
        try
        {
            LocalDevice local = LocalDevice.getLocalDevice();
            local.setDiscoverable(DiscoveryAgent.GIAC);
            server = (StreamConnectionNotifier) Connector.open("btspp://localhost:" + UUID_STRING + ";authorize=false;authenticate=false;encrypt=false;name=" + SERVICE_NAME);
            Log.p("EchoServer():Connector open!", Log.INFO);
        }
        catch (Exception e)
        {
            Log.p("EchoServer():exception" + e.getMessage(), Log.ERROR);
            return;
        }
    }

    public void run()
    {
        isRunning = true;
        try
        {
            while (isRunning)
            {
                StreamConnection conn = server.acceptAndOpen();
                Log.p("EchoServer.Start():Client Connected", Log.INFO);

                Handler handler = new Handler(conn, event);
                handlers.addElement(handler);
                Thread myThread = new Thread(handler);
                myThread.run();
            }
        }
        catch (IOException e)
        {
            Log.p("EchoServer():exception" + e.getMessage(), Log.ERROR);
        }
    }

    public void closeDown()
    {
        if (isRunning)
        {
            isRunning = false;
            try
            {
                Log.p("EchoServer.closeDown():closing", Log.INFO);
                server.close();
            }
            catch (IOException e)
            {
                Log.p("EchoServer.closeDown():exception" + e, Log.ERROR);
            }

            Handler hand;
            for (int i = 0; i < handlers.size(); i++)
            {
                hand = (Handler) handlers.elementAt(i);
                hand.closeDown();
            }
            handlers.removeAllElements();
        }
    }

    public int getNumberOfConnectedClients()
    {
        Handler hand;
        int count = 0;
        Enumeration e = handlers.elements();
        while (e.hasMoreElements())
        {
            hand = (Handler) e.nextElement();
            if(hand.isRunning())
                count++;
        }

        return count;
    }
}
```

bluetooth/Client.java

```
public class Client
{
    private ServiceRecord servRecord;
    private StreamConnection conn; // for the server
    private Event event;

    private Handler handler;

    public Handler getHandler()
    {
        return handler;
    }

    public Client(ServiceRecord servRecord, Event event)
    {
        this.servRecord = servRecord;
        this.event = event;
    }

    public void connect()
    {
        // get a URL for the service
        String servURL = servRecord.getConnectionURL(ServiceRecord.NOAUTHENTIC
ATE_NOENCRYPT, false);
```

```
        if (servURL != null)
        {
            Log.p("Client.run():Connection url found!", Log.DEBUG);
            try
            {
                // connect to the server, and extract IO streams
                conn = (StreamConnection) Connector.open(servURL);
                Log.p("Client.connect():Connected to server!", Log.INFO);

                handler = new Handler(conn, event);
                Thread myThread = new Thread( handler );
                myThread.run();
            }
            catch (Exception e)
            {
                Log.p("Client.run():exception:" + e.getMessage(), Log.ERROR);

            }
        }
        else
        {
            Log.p("Client.run(): No service found?", Log.WARNING);
        }
    }
}
```

bluetooth/Handler.java

```
public class Handler implements Runnable
{
    private StreamConnection conn; // client connection
    private String clientName;

    private InputStream in;
    private OutputStream out;
    private boolean isRunning;

    private Object state;

    private Event event;

    public Object getState()
    {
        return state;
    }

    public void setState(Object state)
    {
        this.state = state;
    }

    public boolean isRunning()
    {
        return isRunning;
    }

    public static String getDeviceName(RemoteDevice dev)
    {
        String devName;
        try
        {
            devName = dev.getFriendlyName(false);
```

```java
        }
        catch (IOException e)
        {
            devName = "Device ??";
        }
        return devName;
    }

    public Handler(StreamConnection conn, Event event)
    {
        this.event = event;
        this.conn = conn;
        try
        {
            RemoteDevice rd = RemoteDevice.getRemoteDevice(conn);
            clientName = getDeviceName(rd);
            Log.p("Handler():connected " + clientName, Log.INFO);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public void run()
    {
        try
        {
            in = conn.openInputStream();
            out = conn.openOutputStream();
            RemoteDevice remoteDev = RemoteDevice.getRemoteDevice(conn);
            event.clientConnected(remoteDev, this);
            serve();
            event.clientDisconnected(remoteDev);
            closeDown();
        }
        catch (Exception e)
        {
            Log.p("Handler.run():exception" + e.getMessage(), Log.ERROR);
            e.printStackTrace();
        }
    }

    private void serve() throws InterruptedException
    {
        isRunning = true;
        String msg;
        while (isRunning)
        {
            msg = readData();

            if (msg == null)
            {
                isRunning = false;
                break;
            }

            //Log.p(clientName + " says:" + msg, Log.DEBUG);

            event.messageReceived(msg, this);

            //Thread.sleep(100);

```

```java
     }
     Log.p("Handler.server():finished!", Log.INFO);
        }

        private String readData()
        {
            byte[] buffer = new byte[10 * 1024];

            int read;
            int lastIndex = 0;
            try
            {
                do
                {
                    read = in.read(buffer, lastIndex, buffer.length - lastIndex);
                    lastIndex += read;

                    // Logger.Log("TEH.Read():received:" + read + "bytes");

                    if (read < 0)
                    {
                        Log.p("Handler.readData():Excp1?", Log.WARNING);
                        Dialog.show("Handler.readData()", "wft1", "ok", null);
                        Thread.sleep(500);
                    }

                    if (buffer[lastIndex - 1] == (byte) '\n' && buffer[(lastIndex - 2)] == (byte) '\r')
                    {
                        buffer[lastIndex - 1] = buffer[(lastIndex  - 2)] = 0;
                        return new String(buffer, "UTF-8");
                    }

                } while (read > 0);

                Log.p("Handler.readData():Excp2?", Log.WARNING);
                Dialog.show("Handler.readData()", "wft2", "ok", null);
                return null;
            }
            catch (Exception e)
            {
                Log.p("Handler.readData():exception" + e.getMessage(), Log.ERROR);
                Dialog.show("Handler.readData():Ex", e.getMessage(), "ok", null);
                e.printStackTrace();
                return null;
            }
        }

        public boolean sendMessage(String msg)
        {
            try
            {
                msg = msg + "\r\n";
                out.write(msg.getBytes("UTF-8"));
                out.flush();
                return true;
            }
            catch (Exception e)
```

```java
                {
                    Log.p("Handler.sendMessage():exception" + e.getMessage(), Log.ERROR);
                    return false;
                }
            }

            public void closeDown()
            {
                isRunning = false;
                try
                {
                    if (conn != null)
                        conn.close();
                    if (in != null)
                        in.close();
                    if (out != null)
                        out.close();

                }
                catch (IOException e)
                {
                    Log.p("Handler.closeDown():exception" + e.getMessage(), Log.ERROR);
                }
            }
        }
```

logic/BackgroundTask.java

```java
public class BackgroundTask extends Thread
{
    private Server server;
    private ServiceFinder serviceFinder;
    private Client client;

    private StateEvent event;

    private boolean isRunning = true;

    public BackgroundTask(StateEvent event)
    {
        if (event == null)
            this.event = new StateEvent();
        else
            this.event = event;
    }

    private void startListening() throws InterruptedException
    {
        event.stateChanged(StateEvent.LISTENING);

        Event serverEvent = new Event()
        {
            Protocol protocol = new Protocol();

            public void clientConnected(RemoteDevice dev, Handler handler)
            {
                event.stateChanged(StateEvent.COMMUNICATING);
            }

            public void clientDisconnected(RemoteDevice dev)
            {
                event.stateChanged(StateEvent.LISTENING);
            }

            public void messageReceived(String message, Handler handler)
            {
                protocol.processMessage(handler, message);
            }
        };

        server = new Server(serverEvent);
        server.start();

        Thread.sleep(60000);

        int wait = 0;
        while (wait < 6)
        {
            if (server.getNumberOfConnectedClients() > 0)
            {
                wait++;
                //Dialog.show("startListening()", "waiting a little more", "ok", null);
                Thread.sleep(30000);
            }
            else
                break;
        }

        //Dialog.show("startListening()", "server closed!", "ok", null);
        server.closeDown();
    }

    private void startDiscovering() throws InterruptedException, IOException
    {
        event.stateChanged(StateEvent.SEARCHING);

        Event clientEvent = new Event()
        {
            Protocol protocol = new Protocol();

            public void clientDisconnected(RemoteDevice dev)
            {
                event.stateChanged(StateEvent.SEARCHING);
            }

            public void clientConnected(RemoteDevice dev, Handler handler)
            {
                event.stateChanged(StateEvent.COMMUNICATING);
                protocol.init(handler);
            }

            public void messageReceived(String message, Handler handler)
            {
                protocol.processMessage(handler, message);
            }
        };

        serviceFinder = new ServiceFinder(Server.UUID_STRING, Server.SERVICE_NAME);

        int wait = 0;
        while (wait < 4)
        {
            if (!serviceFinder.isSearchDone())
            {
                wait++;
                Thread.sleep(10000);
            }
                   else
                       break;
               }

               if (!serviceFinder.isSearchDone())
               {
                   Log.p("run():finder cancelled", Log.DEBUG);
                   //Dialog.show("bg.startdiscovering()", "finder cancelled", "ok", null);
                   serviceFinder.closeDown();
               }

               if (serviceFinder.getServiceTable().size() > 0)
               {
                   Enumeration e = serviceFinder.getDeviceList().elements();
                   while (e.hasMoreElements())
                   {
                       RemoteDevice dev = (RemoteDevice) e.nextElement();

                       if (CacheManager.isDeviceKnown(dev.getBluetoothAddress()))
                       {
                           //Dialog.show("Attention!", "Device found but cached. skipping..", Dialog.TYPE_ALARM, null, "Ok", null, 3000);
                           continue;
                       }
                       else
                           CacheManager.cacheDevice(dev.getBluetoothAddress());

                       client = new Client((ServiceRecord) serviceFinder.getServiceTable().get(dev.getFriendlyName(false)), clientEvent);
                       client.connect();

                       wait = 0;
                       while (wait < 10)
                       {
                           if (client.getHandler().isRunning())
                           {
                               wait++;
                               Thread.sleep(20000);
                           }
                           else
                               break;
                       }

                       if (client.getHandler().isRunning())
                       {
                           Log.p("run():device connection cancelled", Log.DEBUG);
                           //Dialog.show("bg.startdiscovering()", "device connection cancelled", "ok", null);
                           client.getHandler().closeDown();
                       }
                   }
               }
           }
```

```java

        public final void run()
        {
            while (isRunning)
            {
                try
                {

                    event.stateChanged(StateEvent.SLEEPING);

                    String name;
                    try
                    {
                        name = LocalDevice.getLocalDevice().getFriendlyName();
                    }
                    catch (BluetoothStateException e)
                    {
                        Dialog.show("Error", "Bluetooth is not enabled. Please enable BT and restart agent.", "OK", null);
                        break;
                    }

                    int sleep = Helpers.showRandomInteger(0, 5, 0);
                    Thread.sleep(sleep * 1000);

                    if (isRunning && ConfigurationManager.getConfiguration().getDiscoveryMode() != 2)
                        startListening();

                    if (isRunning && ConfigurationManager.getConfiguration().getDiscoveryMode() != 1)
                        startDiscovering();

                    event.stateChanged(StateEvent.SLEEPING);

                    sleep = Helpers.showRandomInteger(0, 10, 0);
                    Thread.sleep(sleep * 1000);
                }
                catch (javax.bluetooth.BluetoothStateException e)
                {
                    Log.p("BackgroundTask():exception " + e.getMessage(), Log.ERROR);
                    Dialog.show("Error", "Bluetooth system is not active.", "Ok", null);
                    break;
                }
                catch (Exception e)
                {
                    Log.p("BackgroundTask():exception " + e.getMessage(), Log.ERROR);
                    e.printStackTrace();
                }
            }
        }

        public void stop()
        {
            isRunning = false;

            if (server != null)
                server.closeDown();

                if (serviceFinder != null && !serviceFinder.isSearchDone())
                    serviceFinder.closeDown();

                if (client != null && client.getHandler() != null && client.getHandler().isRunning())
                    client.getHandler().closeDown();
            }
        }
```

logic/Protocol.java

```java
public class Protocol
{
    private Profile myProfile;
    private ProfileManager profileManager;
    Hashtable result;

    public Protocol()
    {
        myProfile = ConfigurationManager.getConfiguration().getProfile();
        profileManager = new ProfileManager();
    }

    public void init(Handler handler)
    {
        System.out.println("init");
        StringBuffer sb = new StringBuffer();
        sb.append("helo|");
        sb.append(myProfile.getId());
        sb.append("|");
        sb.append(Helpers.dateToXml(myProfile.getDate()));

        handler.setState(new Profile());
        handler.sendMessage(sb.toString());
    }

    private void profileExchangeFinished(Handler handler)
    {
        Profile remote = (Profile) handler.getState();

        //check if we have stored message
        /*
        String message;
        while((message = MessageQueueManager.dequeue(remote.getId()) )!= null)

            handler.sendMessage("mesg|" + message);
        */
        handler.sendMessage("done");
        remote.setDoneSent(true);

        if(remote.isDoneReceived())
            handler.closeDown();
    }


    // TODO sycn  ?
    public synchronized void processMessage(Handler handler, String message)
    {
        String cmd = message.substring(0, 4);

        System.out.println("cmd:" + cmd);
        //Dialog.show("Protocol.processMessage()", cmd, "ok", null);
        Profile remote = (Profile) handler.getState();

        if (message.length() > 5)
            message = message.substring(5);

        if (cmd.equals("helo"))
        {
            if (handler.getState() == null)
            {
                init(handler);
                remote = (Profile) handler.getState();
            }

            String id = message.substring(0, message.indexOf("|"));
            remote.setId(id);

            Profile prof = profileManager.findProfile(id);

            if (prof != null)
            {
                Date date = Helpers.xmlToDate(message.substring(message.indexOf("|")+1));
                if ((date.getTime() - prof.getDate().getTime()) < 1000)  // fast comparison!
                {
                    //Auth remote user, in case he/she re-requests profile!
                    remote.setAuthroized(true);
                    System.out.println("Profile found, nothing new. returning...");
                    profileExchangeFinished(handler);
                    return;
                }
                else
                {
                    profileManager.deleteProfile(prof);
                }
            }

            handler.sendMessage("reqa");
        }
        else if (cmd.equals("reqa"))
        {
            handler.sendMessage("resa|" + profileManager.serializeAnonymousParts(myProfile));
        }
        else if (cmd.equals("resa"))
        {
            profileManager.fillProfile(message, remote);

            ProfileMatcher matcher = new ProfileMatcher();
            Hashtable result = matcher.compare(myProfile, remote);

            double percentage = Double.parseDouble(result.get("Percentage").toString());

                remote.setPercentage(percentage);
                remote.setPriorty(percentage);

                DiaProfileMatchViewer dialog = new DiaProfileMatchViewer(Display.getInstance().getCurrent(), "Comparison", "Auth", "Decline", 0, result);

                if(dialog.showDialog())
                {
                        System.out.println("remote auth req accp!");
                        remote.setAuthroized(true);
                        handler.sendMessage("reqi");
                    }
                    else
                    {
                        System.out.println("remote auth req declined!");
                        remote.setAuthroized(false);
                        profileExchangeFinished(handler);
                    }
                }
                else if (cmd.equals("reqi"))
                {
                    if (remote.isAuthroized())
                        handler.sendMessage("resi|" + profileManager.serializeIdentityParts(myProfile));
                    else
                        handler.sendMessage("decl");
                }
                else if (cmd.equals("resi"))
                {

                    profileManager.fillProfile(message, remote);

                    profileManager.saveProfile(remote);

                    Dialog.show("Friend Saved!", remote.getName() + " was added to friend list.", Dialog.TYPE_CONFIRMATION, null, "Ok", null, 3000);
                    System.out.println("Profile saved!");

                    profileExchangeFinished(handler);
                }
                else if (cmd.equals("decl"))
                {
                    profileExchangeFinished(handler);
                }
                else if (cmd.equals("done"))
                {
                    remote.setDoneReceived(true);
                    if(remote.isDoneSent())
                        handler.closeDown();
                    else
                        profileExchangeFinished(handler);
                }
                else if (cmd.equals("mesg"))
                {
                    System.out.println(message);
                }

            }
        }
```

logic/ProfileMatcher.java

```java
public class ProfileMatcher
{
    public static final String SEPERATOR = ", ";

    private double totalPoints;
    private double matchPoints;

    public Hashtable compare(Profile p1, Profile p2)
    {
        totalPoints = 0;
        matchPoints = 0;

        Hashtable result = new Hashtable();

        result.put("Activities", findCommons(p1.getActivities(), p2.getActivities(), true, true, true));
        result.put("Interests", findCommons(p1.getInterests(), p2.getInterests(), true, true, true));
        result.put("Music", findCommons(p1.getMusic(), p2.getMusic(), true, true, true));
        result.put("Tv", findCommons(p1.getTv(), p2.getTv(), true, true, true));
        result.put("Movies", findCommons(p1.getMovies(), p2.getMovies(), true, true, true));
        result.put("Books", findCommons(p1.getBooks(), p2.getBooks(), true, true, true));


        result.put("Friends", findCommons(p1.getFriends(), p2.getFriends(), false, false, false));
        result.put("Trade", findCommons(p1.getLookingFor(), p2.getHave(), false, false, false));
        result.put("Services", findCommons(p1.getServicesInterested(), p2.getServicesProvided(), false, false, false));


        double percantage = matchPoints / totalPoints;
        percantage *= 100;

        percantage = Math.ceil(percantage);


        if(percantage > 100)
            percantage = 100;

        result.put("Percentage", String.valueOf(percantage));

        return result;
    }

    private String findCommons(Vector v1, Vector v2, boolean calcCount, boolean ordered, boolean addNotCommon)
    {
        if (v1.size() == 0)
            return "";

        double itemValue = 1;
        double itemIndexBonusValue = 1;
        double indexBonus = 0;

        if (calcCount)
        {
            totalPoints += (v1.size() * itemValue);
            if (ordered)
            {
                totalPoints += (v1.size() * itemIndexBonusValue);
                //indexBonus = (v1.size() * itemIndexBonusValue) / ((v1.size() * (v1.size() + 1)) / 2);
                indexBonus = (itemIndexBonusValue * 2) / (v1.size() + 1);
            }
        }
```

```java

        StringBuffer sb = new StringBuffer();

        Enumeration e = v2.elements();
        while (e.hasMoreElements())
        {
            String activity = (String) e.nextElement();

            if (v1.contains(activity))
            {
                if (calcCount)
                {
                    matchPoints += itemValue;
                    if (ordered)
                    {
                        matchPoints += (v1.size() - v1.indexOf(activity)) * indexBonus;
                    }
                }

                sb.append("<b>");
                sb.append(activity);
                sb.append("</b>");
                sb.append(", ");
            }
            else if(addNotCommon)
            {
                sb.append(activity);
                sb.append(", ");
            }
        }

        if (sb.length() > 2)
            return sb.toString().substring(0, sb.length() - 2);
        else
            return "";
    }
}
```

logic/Protocol.java

```java
public class Protocol
{
    private Profile myProfile;
    private ProfileManager profileManager;
    Hashtable result;

    public Protocol()
    {
        myProfile = ConfigurationManager.getConfiguration().getProfile();
        profileManager = new ProfileManager();
    }

    public void init(Handler handler)
    {
        System.out.println("init");
        StringBuffer sb = new StringBuffer();
        sb.append("helo|");
        sb.append(myProfile.getId());
        sb.append("|");
        sb.append(Helpers.dateToXml(myProfile.getDate()));

        handler.setState(new Profile());
        handler.sendMessage(sb.toString());
    }

    private void profileExchangeFinished(Handler handler)
    {
        Profile remote = (Profile) handler.getState();

        //check if we have stored message
        /*
        String message;
        while((message = MessageQueueManager.dequeue(remote.getId()) )!= null)

            handler.sendMessage("mesg|" + message);
        */
        handler.sendMessage("done");
        remote.setDoneSent(true);

        if(remote.isDoneReceived())
            handler.closeDown();
    }


    // TODO sycn  ?
    public synchronized void processMessage(Handler handler, String message)
    {
        String cmd = message.substring(0, 4);

        System.out.println("cmd:" + cmd);
        //Dialog.show("Protocol.processMessage()", cmd, "ok", null);
        Profile remote = (Profile) handler.getState();

        if (message.length() > 5)
            message = message.substring(5);

        if (cmd.equals("helo"))
        {
            if (handler.getState() == null)
            {
                init(handler);
                remote = (Profile) handler.getState();
            }

            String id = message.substring(0, message.indexOf("|"));
            remote.setId(id);

            Profile prof = profileManager.findProfile(id);

            if (prof != null)
            {
                Date date = Helpers.xmlToDate(message.substring(message.indexOf("|")+1));
                if ((date.getTime() - prof.getDate().getTime()) < 1000)  // fast comparison!
                {
                    //Auth remote user, in case he/she re-requests profile!
                    remote.setAuthroized(true);
                    System.out.println("Profile found, nothing new. returning...");
                    profileExchangeFinished(handler);
                    return;
                }
                else
                {
                    profileManager.deleteProfile(prof);
                }
            }

            handler.sendMessage("reqa");
        }
        else if (cmd.equals("reqa"))
        {
            handler.sendMessage("resa|" + profileManager.serializeAnonymousParts(myProfile));
        }
        else if (cmd.equals("resa"))
        {
            profileManager.fillProfile(message, remote);

            ProfileMatcher matcher = new ProfileMatcher();
            Hashtable result = matcher.compare(myProfile, remote);

            double percentage = Double.parseDouble(result.get("Percentage").toString());

                   remote.setPercentage(percentage);
                   remote.setPriorty(percentage);

                   DiaProfileMatchViewer dialog = new DiaProfileMatchViewer(Display.getInstance().getCurrent(), "Comparison", "Auth", "Decline", 0, result);

                   if(dialog.showDialog())
                   {
                       System.out.println("remote auth req accp!");
                       remote.setAuthroized(true);
                       handler.sendMessage("reqi");
                   }
                   else
                   {
                       System.out.println("remote auth req declined!");
                       remote.setAuthroized(false);
                       profileExchangeFinished(handler);
                   }
               }
               else if (cmd.equals("reqi"))
               {
                   if (remote.isAuthroized())
                       handler.sendMessage("resi|" + profileManager.serializeIdentityParts(myProfile));
                   else
                       handler.sendMessage("decl");
               }
               else if (cmd.equals("resi"))
               {

                   profileManager.fillProfile(message, remote);

                   profileManager.saveProfile(remote);

                   Dialog.show("Friend Saved!", remote.getName() + " was added to friend list.", Dialog.TYPE_CONFIRMATION, null, "Ok", null, 3000);
                   System.out.println("Profile saved!");

                   profileExchangeFinished(handler);
               }
               else if (cmd.equals("decl"))
               {
                   profileExchangeFinished(handler);
               }
               else if (cmd.equals("done"))
               {
                   remote.setDoneReceived(true);
                   if(remote.isDoneSent())
                       handler.closeDown();
                   else
                       profileExchangeFinished(handler);
               }
               else if (cmd.equals("mesg"))
               {
                   System.out.println(message);
               }

           }
       }
```

logic/StateEvent.java

```java
public class StateEvent
{
    public static final int LISTENING = 0;
    public static final int SEARCHING = 1;
    public static final int COMMUNICATING = 2;
    public static final int SLEEPING = 3;

    public void stateChanged(int state)
    {

    }
}
```

model/Profile.java

```java
public class Profile implements Persistable, Comparable
{
    private String id = "";
    private String fid = "";
    private String name = "";
    private String birthday = "";
    private String sex = "";
    private String email = "";
    private Vector friends;
    private Vector friendNames;
    private Vector events;
    private Vector eventNames;

    private Vector activities;
    private Vector interests;
    private Vector music;
    private Vector tv;
    private Vector movies;
    private Vector books;

    private Vector have;
    private Vector lookingFor;
    private Vector servicesInterested;
    private Vector servicesProvided;

    private Date date;
    private String signatureValue;

    private byte[] picture;

    private double percentage;
    private double priorty;

    private transient boolean authroized;
    private transient boolean doneReceived;
    private transient boolean doneSent;


    // getters & Setters

}
```

util/HttpDownloader.java

```java
public class HttpDownloader
{
    public byte[] FetchUrl(String url)
    {
        HttpConnection conn = null;
        InputStream stream = null;
        byte[] data = null;
        try
        {
            conn = (HttpConnection) Connector.open(url);
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Connection", "close");

            System.out.println("Status Line Code: " + conn.getResponseCode());
            System.out.println("Status Line Message: " + conn.getResponseMessage());
            if (conn.getResponseCode() == HttpConnection.HTTP_OK)
            {
                stream = conn.openInputStream();
                int length = (int) conn.getLength();
                System.out.println("Content Length:" + length);
                if (length != -1)
                {
                    data = new byte[length];
                    stream.read(data);
                }
            }
            else
            {
                return null;
            }

        }
        catch (Exception e)
        {
            e.printStackTrace();
            return null;
        }

        finally
        {
            try
            {
                if (stream != null)
                {
                    stream.close();
                }
                if (conn != null)
                {
                    conn.close();
                }
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
        return data;
    }
}
```

Program.java

```java
public class Program extends MIDlet
{
    private static MIDlet midlet;

    public static void setMidlet(MIDlet midlet)
    {
        Program.midlet = midlet;
    }

    public static MIDlet getMidlet()
    {
        return midlet;
    }

    public Program()
    {
    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException
    {
    }

    protected void pauseApp()
    {
    }

    protected void startApp() throws MIDletStateChangeException
    {
        try
        {
            setMidlet(this);
            //Log.setLevel(Log.INFO);
            Log.p("Application Stared", Log.INFO);

            //init the LWUIT Display
            Display.init(this);
            try
            {
                Resources r = Resources.open("/gb.res");
                Hashtable theme = r.getTheme(r.getThemeResourceNames()[0]);
                UIManager.getInstance().setThemeProps(theme);
            }
            catch (java.io.IOException e)
            {
                e.printStackTrace();
            }

            Configuration config = ConfigurationManager.getConfiguration();

            if (config == null)
            {
                Log.p("First Run", Log.INFO);
                FrmProfileDownloader welcome = new FrmProfileDownloader();
                welcome.show();
            }
            else
            {
                Log.p("Normal Run", Log.INFO);
                FrmProfileViewer main = new FrmProfileViewer(config.getProfile(), "");
                main.show();
            }
        }
        catch (Exception e)
        {
            System.out.println("Fatal Exception!");
            e.printStackTrace();
        }
    }
}
```