TAMPEREEN TEKNILLINEN YLIOPISTO

Saku Rautiainen

# Prototype for Automatic Content Discovery in Mobile Ad Hoc Networks

Master of Science Thesis

Examiners:
Adjunct prof. Marko Hännikäinen
Dr. Tech. Jukka Suhonen
Examiners and topic approved by the
Faculty Council of the Faculty of
on 3.10.2012

# PREFACE

It has been my pleasure to participate in the social index engine(SoInx) project, to be part of creating something new and extraordinary in the field of social applications. To be able to create something that has the potential to connect thousands of people and help them find common ground.

This thesis was written for the Department of Pervasive Computing at Tampere University of Technology(TUT) as part of SoInx project. This project and my thesis would not have had happened without my fellow coworkers at TUT and Nokia Research Center personnel at Espoo. I would like to thank Adjunct prof. Marko Hännikäinen and Dr. Tech. Jukka Suhonen for their advice regarding writing this thesis. I would like to thank my parents, Arja and Antti Rautiainen, for their constant support and belief that I would one day finish my masters thesis and hopefully graduate. Finally I would like to thank my brother Oskari and my sister Kaisa for being supportive siblings.

# ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Telecommunications Electronics Technology
**Saku Rautiainen: Prototype for Automatic Content Discovery in Mobile Ad Hoc Networks**
Master of Science Thesis, 62 pages
October 2013
Major: Embedded Systems
Examiner:
Adjunct prof. Marko Hännikäinen
Dr. Tech. Jukka Suhonen
Keywords: ad hoc, mobile, social application, content discovery

With growing number of social media websites and people using them, the amount of personal information on the internet is staggering. Using this information we can analyze people's behavior and suggest new likes or friends.

This thesis presents the prototype called SoInx that was implemented with Python and Qt on Nokia N9. SoInx was created to find common interests between users, thus help them find new likes and friends. SoInx does this by taking all the personal data found on a mobile phone and users' social media account, and creating interests from that data. These interests are combined with a salt, salt being device real time clock and hashed before application starts throwing them to the ad hoc network. In cryptography, salt is random data that is given as additional input to a hashing function. By adding salt we increase the security of the hash. Thanks to shared secret encryption method, user privacy is maintained and only those interests are revealed which the other user already knows through having said interests.

In the second part, this thesis presents the user tests conducted with SoInx and their results. User tests were implemented in simulated environment, with 10 users. Each tester conducted his or her own test independent of the others. These user testers were recruited from the university and represented typical active social media users. Results showed that most users thought that finding new people was the most interesting concept in SoInx. Some of the testers found that SoInx displayed too much data at times on the screen. Refining learning algorithm would decrease the amount of useless information on the screen, since then SoInx would filter away more not interesting interests.

# TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietoliikenne-elektroniikan koulutusohjelma
**Saku Rautiainen: Prototyyppi Sisällön Löytämiseksi Automaattisesti Mobiilista Ad Hoc Verkosta**
Diplomityö, 62 sivua
Lokakuu 2013
Pääaine: Sulautetut järjestelmät
Tarkastajat:
Dosentti Marko Hännikäinen
TkT. Jukka Suhonen
Avainsanat: ad hoc, mobiili, sosiaalinen aplikaatio, sisällön löytäminen

Sosiaalisen median palveluiden sekä käyttäjämäärien kasvaessa internetistä löytyvä henkilökohtainen tieto ihmisistä on valtaisa. Tätä tietoa hyväksi käyttämällä pystytään analysoimaan ihmisten käyttäytymistä ja ehdottamaan uusia tuttavuuksia tai kiinnostuksen kohteita.

Tämä diplomityö esittelee prototyypin(SoInx), joka on luotu käyttäen Pythonia ja Qt:ta Nokian N9 puhelimelle. SoInxin tarkoitus on löytää käyttäjien väliltä yhteisiä tykkäämisiä, tämä auttaa käyttäjiä löytämään uusia tykkäämisiä ja kavereita. SoInx louhii kännykästä ja käyttäjän antamasta sosiaalisen median tilistä tietoja, luoden niistä kiinnostuskohteita. Nämä kiinnostuskohteet salataan lisäämällä niihin suola, suolan lähteenä toimii puhelimen kello. Suola on kryptografiassa satunnaista dataa, joka annetaan hajautusfunktiolle. Funktio lisää suolan dataan ja parantaa datan hajautusta. Koska käytetään jaettu salaisuus todennusta, käyttäjän yksityisyys ei missään vaiheessa rikkoudu vaan ainoastaan ne asiat paljastetaan, jotka toinen käyttäjä jo tietää.

Viimeisenä tämä diplomityö esittelee prototyypille tehdyt käyttäjätestit ja niiden tulokset. Käyttäjätestit toteutettiin simuloidussa ympäristössä, 10:llä eri henkilöllä. Jokainen testaaja suoritti testin itsenäisesti muista riippumatta. Testaajat rekrytoitiin yliopistolta ja jokainen heistä edusti tyypillistä nuorta aktiivista sosiaalisen median käyttäjää. Testin tulokset osoittivat, että suurin osa käyttäjistä piti siitä ideasta, että SoInx auttaisi heitä löytämään uusia tuttavia. Osa käyttäjistä oli sitä mieltä, että laitteen näytöllä oli välillä liikaa asioita esitettävänä välillä. Oppimisalgoritmin säätäminen mahdollistaisi sen, että SoInx suodattaisi pois sellaiset kiinnostuksenkohteet, jotka ovat vähemmän kiinnostavia.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# TERMS AND DEFINITIONS

| | |
|---|---|
| 3G: | Third generation of digital cellular network technologies |
| Ad hoc: | Solution designed for specific task |
| API: | Application programming interface is a specification of how software components act |
| CPU: | Central processing unit, carries out the instructions of a computer program |
| C++: | An object oriented programming language |
| Edge: | 2G mobile phone technology that allows faster data transmissions than standard GSM |
| GPRS: | Packet oriented mobile data service in 2G and 3G cellular networks |
| GSM: | Standard that describes protocols for 2g digital cellular networks |
| HOP: | Term used to describe traffic between two connected nodes on ad hoc network |
| HOP count: | Number of hops it takes for the packet to reach its destination |
| Interpreter: | Computer program that performs instructions written in programming language |
| Linux: | Unix-like operating system |
| LOC: | Lines of code, software metric used to measure the size of a computer program |
| Maemo: | Linux based OS created by Nokia Inc. for hand-held computers, predecessor to Meego |
| Meego: | Linux based OS created by Nokia Inc. and Intel Inc. for hand held computers |
| mp3: | Audio format for storing audio data |
| N9: | Nokia N9, mobile handheld computer with a Linux based os, Meego |
| OS: | Operating system, collection of computer programs that manage computer hardware resources and provides services for computer programs |

| | |
|---|---|
| PDF: | Proprietary file format used to display content independent of hardware, OS and software |
| Python: | Dynamically typed programming language that is interpreted |
| PyQt: | Python bindings for Qt framework |
| Qt: | An application framework |
| SoInx: | Abbreviation of the Social Index engine |
| UML: | Modeling language used in software engineering |
| URL: | Uniform Resource Locator, address used to locate information on internet |
| WCMDA: | Air interface standard found in 3G networks |
| WLAN: | Wireless Local Area Network |

# 1. INTRODUCTION

SoInx stands for Social Index engine and it is an application meant to find connections between users using their personal data, such as likes, friends, favorite music etc. Enabling users to likes common to them would enable them to make new friends and acquaintances.

In every network there exists a lot of content, and identifying the content pertinent to the user is very important. Automatic content discovery is about discovering content which is important to the user and learning from the interests and actions of the user. This learning enables the program to show which pieces of content, interests, are most interesting.

Ad hoc networks are self-organizing and adaptive [1], which enables them to form a network without previous existing infrastructure anywhere where there are two or more devices and to maintain the networks even if some devices leave. If they are not in range of each other, intermediate nodes must be used for relaying. Ad hoc Wireless Local Area Networks (WLAN) use different WLAN devices, allowing communications over long distances.

Since ad hoc networks can be formed on the fly without existing infrastructure, it makes them very usefull in places where there is not stable internet connection or no internet connection at all. Third world countries are such places where access to internet is limited and ad hoc networks make it possible for people to have stable and reliable networks between devices for exchanging information, pictures etc. In addition ad hoc networks provides very good privacy since there are no servers handling transmission between starting and ending point

Mobile devices using SoInx would form an ad hoc WLAN network to communicate between devices. With this concept SoInx could be used everywhere in the world where there are two or more devices. Ad hoc networks also gives SoInx the capability to connect people who are close to each other by finding relations between people who are physically close.

Figure 1.1 displays an ad hoc network constructed with five mobile devices, named *A*, *B*, *C*, *D* and *E*.

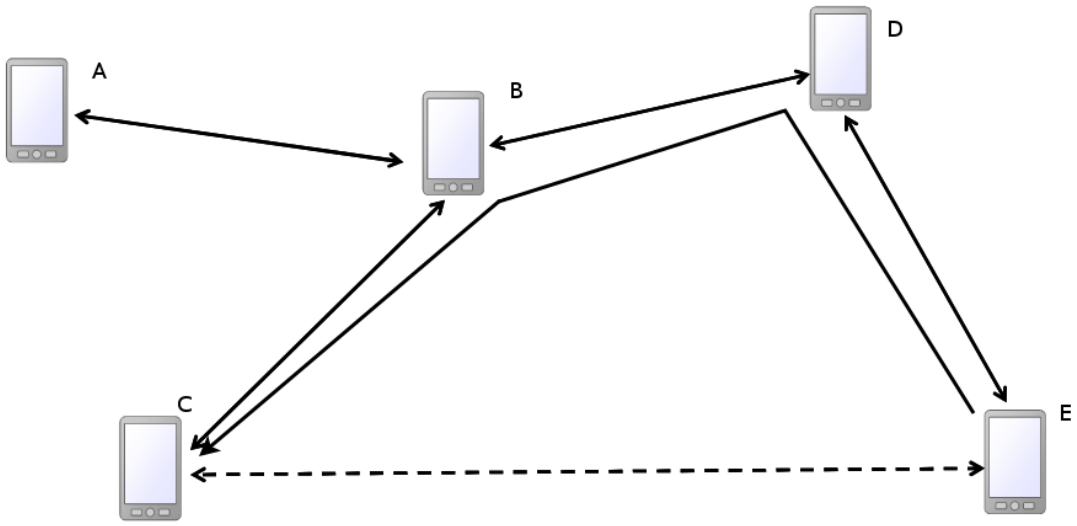


Figure 1.1: Mobile ad hoc network. For message from device $E$ to pass onto device $C$, devices $B$ and $D$ must pass it through them since $E$ and $C$ are not in rage of each other.

## 1.1 Background

This thesis presents the design for a SoInx user interface (UI), use cases and user tests run on SoInx. SoInx is a co-operative development with Nokia Research Center and it was developed as a prototype for the Nokia Instant Community platform. This thesis was created as a result of the social index project at Tampere University of Technology, first at Department of Computer Systems and then later on the same project continued under Department of Pervasive Computing. The project was begun in early 2012. It is continuation of the Twin project that was began in 2009. Twin was a social mobile application that facilitated chatting, exchanging files and forming communities in wireless ad hoc network.

SoInx prototype was developed for the mobile Nokia N9 device and desktop use, with N9 running a Linux operating system(OS) called Meego and desktop running a Linux OS called Ubuntu.

The work of the author for this thesis consisted of making user interface prototypes in co-operation with other developers on the team. Designing, implementing and analyzing user tests was mostly the responsibility of the author as well. The graphical design for SoInx UI came from other project members. The plug-in architecture for interest inputs and SoInx core was implemented by other developers. The SoInx simulator that was used for testing in thesis was implemented previously [2]. Article about SoInx, privacy in it and the results of the user tests was published

and presented at workshop on Mobile Cloud and Social Perspectives(MoCSoP) in September 2013 [3]. Privacy in SoInx is touched upon more on the upcoming Master thesis by Mikko Vataja. Mikko Vataja was also a member of the SoInx research team.

## 1.2 Structure of thesis

The structure of this thesis is as follows. The requirements and constraints for prototyping are discussed in Chapter Requirements and constraints for prototype. Related work in the field of mobile social applications is presented in Chapter Related work. Inputs for content in SoInx are discussed in Chapter Inputs for content. Use cases for SoInx are presented in Chapter SoInx use cases. General architecture is presented in Chapter Architecture. Prototype implementation is discussed in Chapter SoInx Implementation. User interface prototypes are presented in Chapter User Interface. Testing of the prototype and debugging is discussed in Chapter Testing. Results of user tests are presented in Chapter User tests.

# 2. REQUIREMENTS AND CONSTRAINTS FOR PROTOTYPE

SoInx has multiple requirements and constraints as a result of the selected platform, technologies and test demands. These requirements and constraints are divided into four categories: requirements for UI, requirements for core, constraints of the platform and constraints of the simulation.

## 2.1 UI requirements

The UI requirements for SoInx consist of general and feature requirements. General requirements consist of ease of use and scalability. Feature requirements consist of displaying nearby users and interaction with them.

A major requirement for UI was ease of use; hard to use and non-intuitive UI only serve to alienate users away from SoInx before they realize its full potential.

### 2.1.1 Scalability

SoInx was designed with mobile ad hoc networks in mind, by displaying users who are near each other. This means that SoInx must work in an environment where there are many users near each other, such as a schools or work places.

Items above limit the UI. The UI must be easy to use with large number of people nearby. As an ecample SoInx must be able to must be able to display cleanly all users and how interesting they are without cluttering the display. Displaying interesting people and how interesting they are is a major requirement of the UI while filtering away not interesting people is major requirement for the SoInx evaluation and learning algorithm.

### 2.1.2 Displaying nearby users

Displaying nearby other users and how interesting they are to the primary user is very important, especially if one is using SoInx in a place where there are lots of

other users with SoInx. For each user the UI would have to distinguish the important and interesting people from the crowd. If a user only sees non interesting or cannot differentiate between interesting and not interesting people on the display then his interest in SoInx will diminish.

### 2.1.3 Interaction with others

As a social application, SoInx was required to have a way of interacting with interesting people. To enable this interaction, liking another user and sharing a contact card methods were required. Like was invented as a lesser method of social contact, one that simply requires the least effort on the part of user to show interest. This would show up as Like on the other users device. Sending contact card method is a more involved social contact method, user must actually reveal someting about himself. This meant implementing these functionalities into the UI in a way that makes them intuitive and easy to use.

## 2.2 Core requirements

Having the ability to add more inputs for SoInx, inputs being sources of interests, after designing and implementing the core was a major requirement. This is especially important because social media networks are known to come and go quickly. Being able to easily add new social media sites and new sources of information enables longer lifetime for the application with minimal changes.

## 2.3 Constraints of Platform

SoInx has two target platforms, desktop PC for simulation purposes and N9 for actual mobile testing. Linux OS was decided as the common factor, since it can be found in both desktop computers and mobile hand-held computers. N9 was chosen as mobile platform because it was one of the few modern mobile devices with Linux OS, WLAN and touch screen. While there many modern mobile phones and Android is linux bases smart phone, Android is not true Linux in the sense that development tools are very limited and that you cannot easily compile your own tools for Android. N9 comes with terminal installed and standard set of Linux development tools [4].

Running simulation with hundreds of SoInx instances on desktop computer places constraints on central processing unit(CPU) and memory usage. It is important to note that CPU and memory also pose constraints in the N9 due to its smaller computation capability. Also the WLAN in N9 limited SoInx on the bandwidth

Table 2.1: Specifications of the N9

| Feature | Value |
|---|---|
| Dimensions | 116.4 x 61.2 x 12.1 mm |
| Weight | 135 g |
| Display | 3.9 inch AMOLED touch screen |
| Resolution | 480 x 854 pixels |
| Permanent memory | 16GB or 64GB flash, not extendable with micro SD |
| Application memory | 1 GB |
| Processor | ARM Cortex A8 1000 MHz |
| Keyboard | Virtual QWERTY keyboard |
| Network | GSM, GPRS, EDGE, WCMDA, HSPA, Bluetooth 2.1, WLAN b/g/n, FNC |
| Network security | WPA2 (AES/TKIP), WPA, WEP |
| Camera | 7 Mb touch focus, 720p video |
| Battery | 3.7V 1450mAh |
| Qt | Qt 4.7. preinstalled |
| Qt Mobility | Qt Mobility 1.2 preinstalled |
| Other API's | OpenGL ES 1.1, OpenFL ES 2.0 |

and protocol. Finally, the touch screen display poses constraints due to a need to operate the application easily with fingers while displaying information clearly.

### 2.3.1 Harmattan version of Meego platform on N9 hand-held computer

Harmattan is a version of the Meego OS for hand-held computers that Nokia created. Unlike other versions of Meego, Harmattan is still very much based Debian GNU/Linux distribution and uses apt-get packaging manager [4]. The only devices using Harmattan are the Nokia N9 and the Nokia N950. Harmattan was a continued development of the Maemo OS, used in previous Nokia N series hand-held computers, N710, N800, N810 and N900 [5]. While official designation for the OS in N9 is Meego 1.2 Harmattan, it has been called Maemo 6 device [6].

Meego while not as flexible as Maemo, is still much more flexible than most other mobile OS's. Almost all programming languages that are available for Linux are also available for Meego. Official UI development tool for Meego is QML, while C, `C++` and Python are also supported [4]. PySide packages exist for Meego, but they are somewhat buggy whereas the more mature PyQt is not available in the repositories [7] [8].

N9 specifications are shown in table 2.1 [4].

### 2.3.2 Touch screen interface

N9 has a 3.9 inch touch screen with a resolution of 480x854 pixels. User input is given through tapping the screen [4]. This places some constraints on how the UI works. For example, buttons, menus etc. must be large enough to be used with fingers without taking up too much space on the screen. Also relevant information must be shown in a fashion that can be seen on a small screen.

### 2.3.3 User interface constraints

Meego 1.2 Harmattan has interface guidelines given on Harmattan UX guidelines [9]. While these specify QML components that are available, they also specify how application UI should be constructed. QML is a tool for developing user interfaces for Qt applications. UIs are constructed in QML by defining color, size etc. of available UI components [10].

Each window in Harmattan always has a toolbar at the bottom. Windows take up the whole screen and stack on top of each other. Moving back to a previous window is achieved by tapping back button; the back button closes the current window. Figure 2.1 displays how movement between different user views is done in N9.

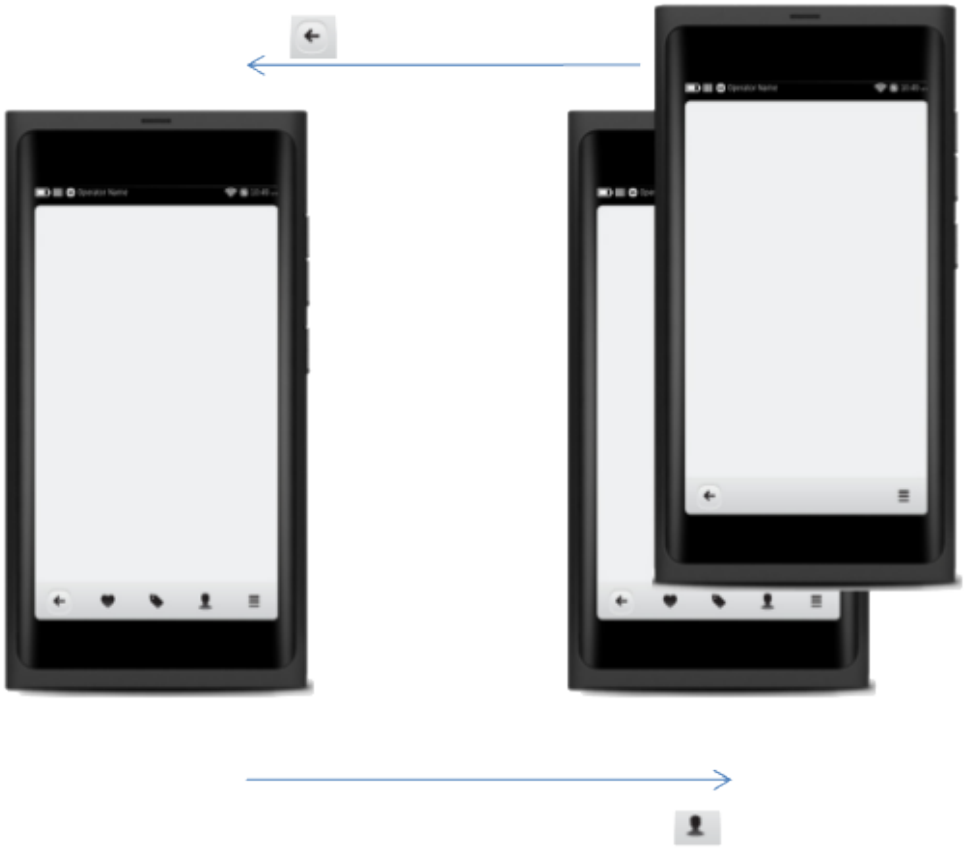

Figure 2.1: Second UI view is opened on top of the first view by clicking on a button in the toolbar. This second view is closed by using the back button, arrow icon.

## 2.4 Simulation constraints

Verifying usability of SoInx and its learning algorithms would require a massive simulation or a lot of user tests. Simulation placed constraints on SoInx since the simulation machine is a normal desktop computer, albeit with 16 GB of ram and latest quad core CPU. This meant minimizing memory footprint and CPU usage while simulating SoInx.

### 2.4.1 Memory usage

There is finite amount of memory on the simulator PC when it is not running simulations, running hundreds of SoInx applications would mean even less memory. This meant that UI would have to be separated from core so that the SoInx application could function without UI during simulation. This would decrease the memory usage per SoInx application and allow for larger simulation.

### 2.4.2 CPU usage

A major constraint for SoInx is the CPU usage. CPU usage is critical on a PC where many SoInxes would be running on the same machine. This meant that anything not needed during simulation or user tests, such as encryption, communication between dummy SoInx applications, etc. would have to be constructed in such a way that disabling them is easy. This would lessen the load on CPU and allow for faster simulations.

# 3. RELATED WORK

Number of research papers and applications have been made that attempt to connect people with each other using content that matches with both users.

## 3.1 Related mobile social applications

Nathan Eagle and Alex Pentland of MIT Media Laboratory introduced an automatic content discovery application called Serendipity in their article *Mobile Matchmaking: Proximity Sensing and Cueing*, that uses Bluetooth for communication between devices. Serendipity is a match making software that allows users to make profiles and describe the kind of attributes they are looking for in a person. These user attributes are given weights in Serendipity, using these weights Serendipity calculates a score for users who are nearby, and if their score reaches a threshold it alerts the user to the fact that there is a user nearby who matches [11].

Per Persson, Jan Blom and Younghee Jung of Nokia inc. discuss an application called DigiDress in the article *DigiDress: A Field Trial of an Expressive Social Proximity Application*. DigiDress allows users to create profiles, in the article named identity expressions. These profiles contain information about the user, such as name, occupation, favorite things, interests in other words. Devices with DigiDress would communicate to each other using Bluetooth and allow the user to see other devices in the area with DigiDress.

The DigiDress application could on its own look around and fetch other peoples DigiDresses. When left to run on its own the user would be notified by a sound signal whenever another device using DigiDress was nearby [12].

Soulakshmee D. Nagowah of University of Mauritius proposed in the article *Aiding social interaction via a mobile peer to peer network* to use Bluetooth technology in an application, BDATE, for automatic content discovery. BDATE is an application that aims to silently communicate to other nearby devices information of the user and what the user is looking for to facilitate match making. Once a match is detected the user is notified via the application. BDATE was successfully tested in Nokia S60 series phones and Motorola phones [13].

Ben Dodson, Ian Vo, T.J. Purtell, Aemon Cannon and Monica S. Lam of Stanford University present Musubi, a mobile application for sharing data between users in their article *Musubi: Disintermediated Interactive Social Feeds for Mobile Devices*. Musubi uses a central server and Internet to exchange data instead of user to user communication due to security concerns. Users can friend each other, from groups, share media and chat [14].

Eric Paulos and Elizabeth Goodman of Intel Research talk about familiar Strangers in their article *The Familiar Stranger: Anxiety, Comfort, and Play in Public Places*. These familiar strangers are people you meet every day but do not interact with. For example when taking the bus to work in the morning you might meet the same person every day. In their article they proposed the use of Jabberwocky devices, these devices use 'digital scent', everyday devices which send low powered radio or wireless protocols, such as Bluetooth. Over time Jabberwocky logs each person you come into contact with repeatedly [15].

Twin is an open source mobile application developed by the Department of Computer systems at Tampere University of Technology and its social networking was based on proximity to other devices. Devices with Twin would form ad hoc WLAN networks and then could communicate with each other. Twin would show nearby users and you could chat with them or send files. Twin also allowed users to form communities [16][17]. Figure 3.1 displays screenshot of Twin.


Figure 3.1: Screenshot of Twin.

Table 3.1: Automatic content discovery applications.

| Name | Year | Central servers | Network layer | Learning | Platform | Implementation Technology |
|---|---|---|---|---|---|---|
| Serendipity [11] | 2005 | Yes | Bluetooth, Internet | No | Nokia S60 | Not mentioned |
| DigiDress [12] | 2005 | Yes | Bluetooth | No | Nokia S60 | Not mentioned |
| Musubi [14] | 2011 | Yes | Internet | No | Android | Dalvik |
| BDATE [13] | 2010 | No | Bluetooth | No | Nokia S60 and Nokia N and Motorola | Java |
| Jabberwocky [15] | 2004 | No | Bluetooth | No | Not mentioned | Not mentioned |
| Twin [16][3] | 2012 | No | ad hoc WLAN | No | Nokia N900 | Python |

## 3.2 Summary

Serendipity [11], DigiDress [12], Jabberwocky [15], and BDate [13] use Bluetooth as their network layer.

Serendipity, DigiDress, and BDate allow user to create profiles, using these profiles these applications communicate with other devices and exchange information. They would alert the user when someone else is nearby that matches with them.

Serendipity, Musubi [14] and DigiDress utilize central servers.

Musubi, and Twin [16][3] both allowed for chatting between users.

Serendipity, DigiDress, and BDATE used Nokia S60 devices, as well as Bluetooth. Their articles did not mention implementation tool, but since S60 was the platform it limits choices to Java and Symbian `C++` [18]. Qt and Web Runtime are much later addition to Symbian developer tools, thus eliminating them from choice of implementation tools.

Social networking in mobile devices has been studied before, as the applications in table 3.1 prove. But none of them have combined mobile ad hoc network, different social medias and learning into one working application.

# 4. INPUTS FOR CONTENT

People are connected by sharin interests such as music, books, friends etc. SoInx aims to use these common things to connect people.

With multitude of Social media websites, such as Facebook [19], and personal data on phone, we have number of potential inputs for SoInx from which to collect interests. Table 4.4 holds all the input sources that were considered and whether or not they were implemented for SoInx.

## 4.1 Social networks as interests

Most social media sites use some sort of application programming interface (API) that enables applications to access data they hold. This usually requires application to identify with the site using user name and password. Using these APIs user data could be accessed from site, user data such as friends, locations, likes etc.

Facebook input was the first social media site input that was created for SoInx. This was because of the popularity of Facebook and the amount of interests that could be collected from there. Facebook API uses OAuth 2.0 for authenticating user and granting access to said users information and data [19].

It was decided to make input for Twitter despite there already being Facebook input, because due to the way Twitter works, mining data for SoInx simulations and using said data has less moral issue. All the data on the Twitter is freely avalaible to anyone with internet, where as in Facebook you need an account and to be friends with people. Twitter uses OAuth 1.0 for authentication [20].

LinkedIn input was started but never finished because of lack of time and large number of features that still needed to be implemented. LinkedIn supports both XML and JSON for data format. LinkedIn utilizes similar authentication method as Twitter, OAuth 2.0 [21]. Implementing Google+ support was discussed but never started due to lack of time and disinterest since it would not offer any new content to SoInx.

Table 4.1: Possible interests from social media.

| Type | Description |
|---|---|
| Facebook friend | User's friends in Facebook |
| Facebook like | User's like taken from Facebook |
| Twitter follower or followee | Person whom user follows or who follows user on twitter |
| Twitter tweet | User's Twitter post, called tweet in Twitter |

Interests that SoInx supports for both Facebook and Twitter are presented in table 4.1.

## 4.2 Data on mobile device

Today's mobile devices, smart phones, are as much computers as your everyday desktop computer. Along with that comes wealth of personal data present on these devices. SoInx is capable of finding interests from phones phone book, media and PDF files located on the phone. Even the presence of third party apps can tell something about a person.

Typical Address book holds person's name, phone number, email address or web page url. Making address book into a n input was thus very easy choice, because of the multitude of information available there. Table 4.2 holds all the interests that were implemented from phonebook input. Phonenumber, email address and web page were thought to be common enough that they exist in every address book and would make easy interests to create.

Most media formats support some sort metadata, this metadata includes useful information about the media, be it video or audio. Even some game formats support this. Table 4.3 shows the metadata types found in audio, video and PDF files that were implemented as interests. Instead of PDF files we could have metadata from eBooks etc.

Mobile phones hold also lots of other useful information like cell tower location, GPS location, application names and ID's. Cell tower ID or GPS location with

Table 4.2: Possible interests from address book.

| Type | Description |
|---|---|
| Phone Number | Phone number found in the address book |
| Email address | Email address found in the address book |
| Web page | Web page url |

Table 4.3: Possible interests from media.

| Type | Description |
|---|---|
| Video artist | Interests for the original artist of the video |
| Video genre | Interest for the genre of the video, documentary, comedy etc |
| Video title | Interest for the title of video |
| Audio artist | Interest for the artist performing on the audio file |
| Audio genre | Interest for the genre of the music, rock, classic etc |
| Audio album | Interest for the album in to which the audio file belongs to |
| Audio Title | Interest for the title of the audio file |
| PDF author | Interest for the author of the PDF file |
| PDF title | Interest for the title of the PDF file |

time allows SoInx to compare to other users and determine if someone else was also there. If there are number of coinciding places and times, it becomes interesting. Camera pictures and their metadata, such as timestamps and location data was also considered. The amount of inputs and interests which could be gleamed from said inputs is staggering, but due to lack of time only small amount of them were implemented.

## 4.3 Summary

Number of different inputs is almost endless, only limited by the data the phone possesses. Location was one input that was considered but not implemented. In location cell tower identifiers, WLAN node identifiers, GPS location, etc. could form with time an interests which tells time and place, useful for connecting people who are in the same places, even at the same time. Also applications were considered as sources for inputs, N9 for example supports package name, version and description [22]. This could be useful for connecting people who use the same software.

The inputs that were implemented are presented in table 4.4. These were chosen for their ease of implementation and readily available data. We used inputs to provide data for SoInx, this data formed the basis of comparing users and their likes, interests.

Table 4.4: Sources of interests, inputs.

| Input | Description | Implemented |
|---|---|---|
| Facebook | Friends and likes made into interests | yes |
| Twitter | Tweets, those who follow and those whom are followed are made into interest | yes |
| LinkedIn | Connections could be made into interests | no |
| Google+ | Friends could be made into interests | no |
| Audio files | Artist, title, album etc. meta data is made into interests | yes |
| Video files | Director, title, genre etc. meta data is made into interests | yes |
| PDF files | Author and title is made into interests | yes |
| Application information | Information the phone might have of application installed | no |
| Address book | Names, phone numbers and email addresses are made into interests | yes |
| Keywords | Editor for adding and removing keywords into SoInx. Each keyword is an interests. This was made purely for testing | yes |
| Location | Location based on cell towers or WLAN ID and GPS. Meeting people who visit the same places at the same. | no |

# 5. SOINX USE CASES

While SoInx is a social application, it is most important use case is not every day communication between people. SoInx is meant for finding new people based on common interests. Using this we can share personal information with people we find interesting. If the user does not wish to share information but likes another user because of the common interests, the user can Like the other user with SoInx. Like here being much akin to Facebook's "like".

SoInx itself could support more use cases, such as chatting with people or sharing files, but they were not implemented due to lack of time, or because they already been implemented in Twin and they did not present new challenges or avenues of research.

Following sections present the use cases, with each section having rationale explaining the reasons for the use. There is also description that briefly describes the use. Each section also holds actor list and scenario explaining in what situation the use case might be used in. Figure 5.1 displays all the user cases.

Each section also has a Unified Modeling Language (UML) activity diagram. Rounded rectangles represent actions, while rectangle boxes with underlined text represent data. Ellipses refer to other user cases. Diamonds represent forks in the user action.

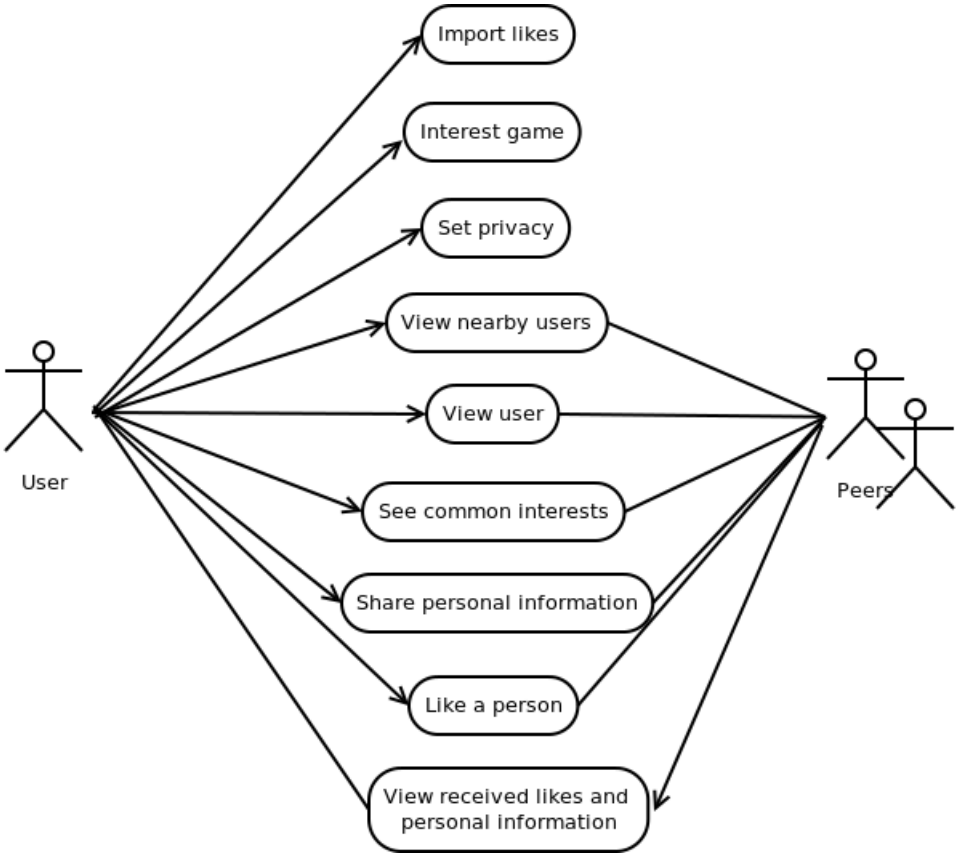


Figure 5.1: Use cases for SoInx. User is the actor group that is using the device with SoInx, while peers are other users, people seen in the network. Arrows headed connection indicates active participation to the use case while regular line connection indicates passive participation in use case. Not all use cases require peers in the network.

## 5.1 Import likes

**Rationale**

Importing likes to SoInx from various inputs is necessary for SoInx to work and it is very important feature in SoInx.

**Description**

The user needs to import likes to SoInx in order to detect users with similar interests.

**Actors**

User

Peers (passive)

**Scenario**

The user starts SoInx for the first time. SoInx does not have any data and does not show any users nearby because of that. User opens *import likes view*. User chooses Facebook input for importing data from Facebook. Facebook input UI opens. This UI is a SoInx web browser. User is shown Facebook login page and adds his user name and password. User presses sign in and web browser closes. In the background SoInx Facebook input imports more Facebook data and converts it into interests for SoInx.

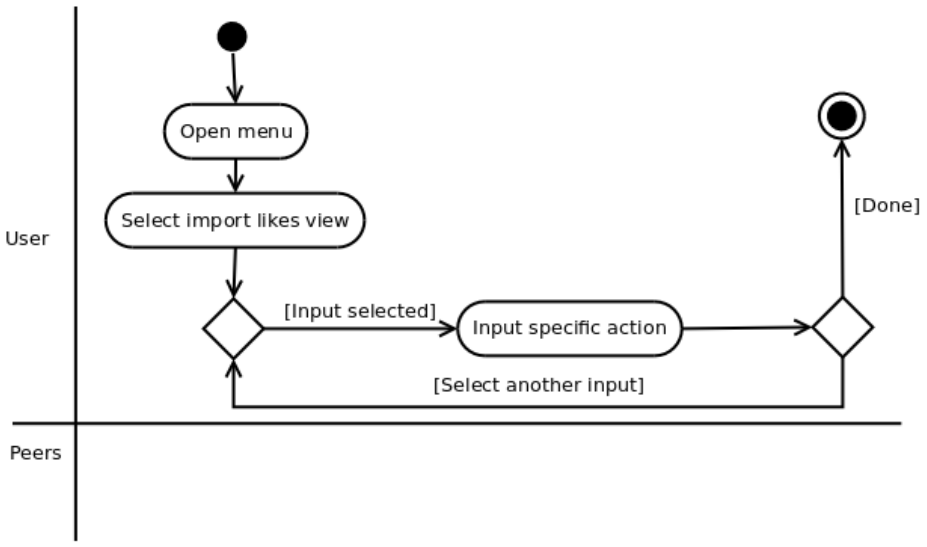


Figure 5.2: Activity diagram for importing likes.

## 5.2 Rate interests

**Rationale**

Calibrating interests is important for the user, this lessens the number of false alerts.

**Description**

The user would like to rate specific interests either interesting or not interesting

**Actors**

User

**Scenario**

After user has started SoInx and imported interests, he will rate interests by starting interest game. Interest game displays those interests to user which had picture associated with them, since picture describes the interests much better than plain text. User will click + or - button to rate interest. In the background the game rates those interests given + sign as more interesting while those that received - sign are rated lower.

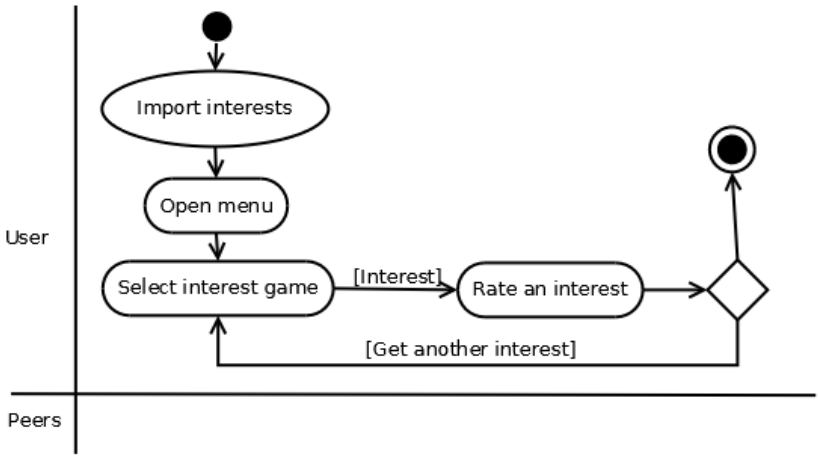


Figure 5.3: Activity diagram for rating interests.

## 5.3 Set privacy

**Rationale**

People want to limit the amount of information they reveal to others. Most people do not wish to reveal sensitive information, such as sexual orientation, religion etc.

**Description**

User selects input and chooses which of the three levels he wishes for the input: hidden, share with those who have same, or share to all.

**Actors**

User

**Scenario**

After having started SoInx, user goes to the *privacy setting view* through menu. In the *privacy setting view* he can set the privacy for different inputs, Facebook, Twitter, mp3's etc.

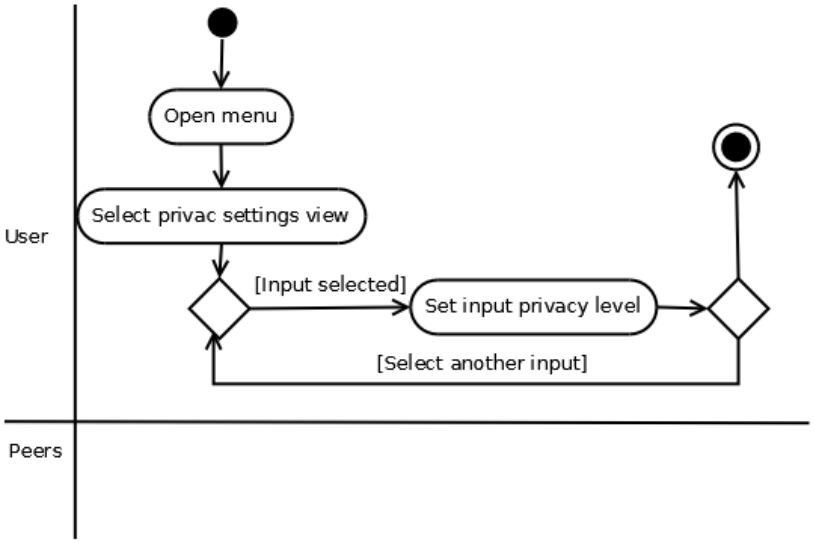


Figure 5.4: Activity diagram for setting the input privacy levels.

## 5.4 View nearby users

**Rationale**

Seeing users near you and what their index and top interests type are is important to the user. It allows differentiating between possibly interesting people and not interesting.

**Description**

The user would like to see people near him and how interesting they are. Additional interest is close they are to the user, distance being denoted by the hops the messages takes.

**Actors**

User

Peers (passive)

**Scenario**

Once user has started SoInx and imported some Interests, SoInx will display nearby users in the radar in *main view*. These users are shown with different colors and icon size depending on how interesting they are to the user.

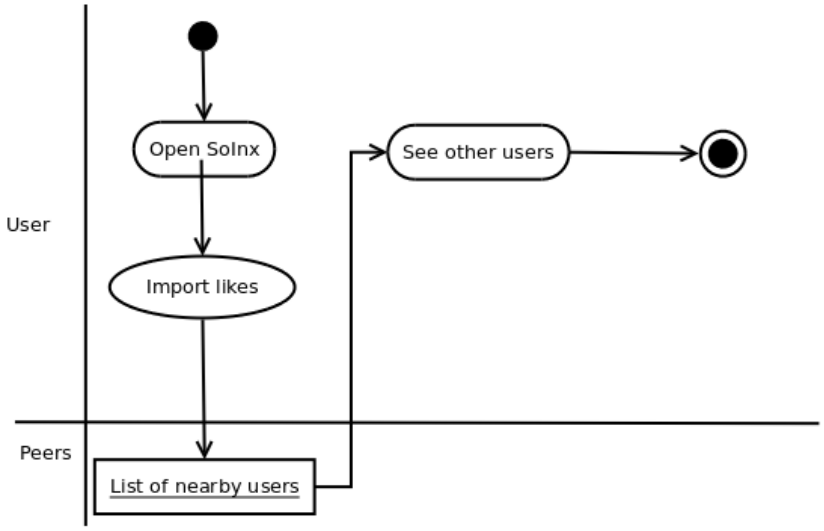


Figure 5.5: Activity diagram for displaying nearby users.

## 5.5 View user

**Rationale**

Seeing who is near you and what his index is, aka how much you have in common, is very important feature in SoInx.

**Description**

The user would like to see a specific other user near him and what his top interest is.

**Actors**

User

Peer (passive)

**Scenario**

After having seen all the nearby users in the radar of the *main view*, he selects one of the users. *User view* pops up from top of the screen. It displays most common interest to the user and several options. These options are like, see more, later and not interesting. In the background clicking one of the buttons will trigger further action and learning. If user clicks *like*, this will be transmitted to the other user and it teaches the system that the top interest was indeed interesting. *See more will* display rest of the common interests whereas *later* will not teach system and will close the view. *Not interesting* will teach the system that the interest was not interesting and closes the view.

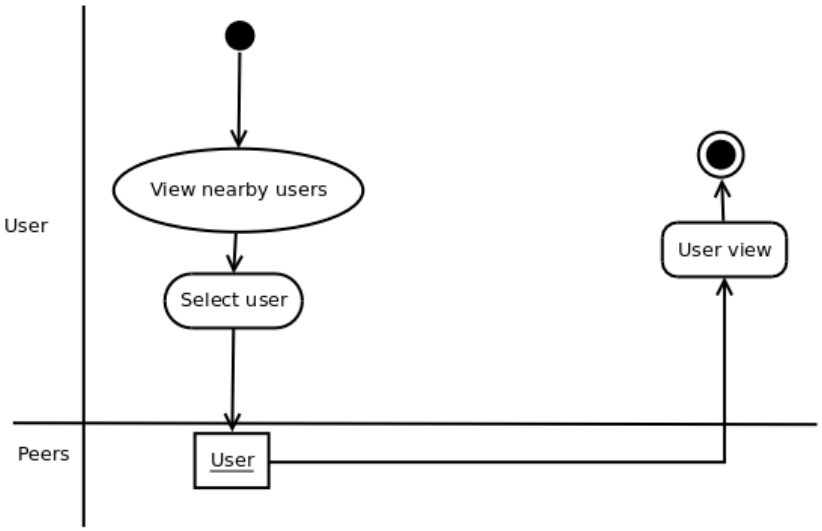


Figure 5.6: Activity diagram for viewing a user.

## 5.6 See common interests with user

**Rationale**

Seeing the common interests between user and other SoInx user is also important feature.

**Description**

The user would like to see common interests he has with another user and how these interests are rated.

**Actors**

User

Peer (passive)

**Scenario**

User is viewing another user has decided to see all the common interests by clicking *See More*. Once user clicks *See More*, SoInx UI pops up *user interests view*. This view shows in a list all the interests that they share. On top of the view is the button *Contact this person*, for opening send *contact card view*. This is used for contacting other users.

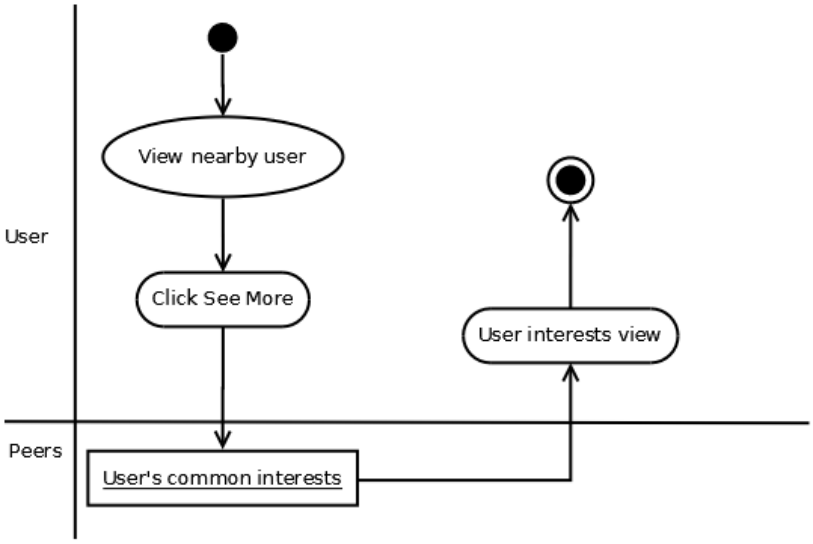


Figure 5.7: Activity diagram for viewing common interests.

## 5.7 Share personal information

**Rationale**

It is natural to share personal information in social networks among common people or friends. Information can be fake or real, it can consist of a number of details.

**Description**

User can add name, email, phone number, nickname etc. into a contact card which he can send to other people.

**Actors**

User

Peers (passive)

**Scenario**

User has found another user with whom he has many interests in common. Most of these interests are extremely important to the user and he wishes to know more about this person. User pushes the *Contact this person button*. Once *send contact card view* opens up user will choose to share his email and nickname with the other user to see if he is interested in further contact.

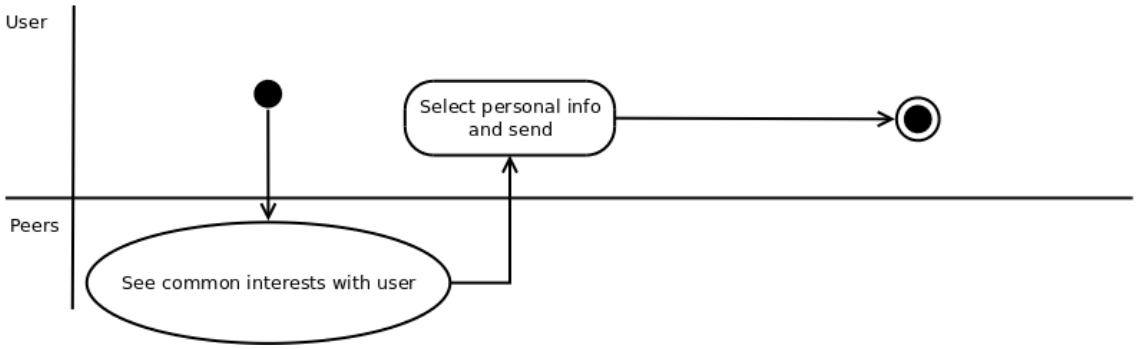


Figure 5.8: Activity diagram for sharing personal information.

## 5.8 Like a person

**Rationale**

The user can like another person for the common interests they share.

**Description**

The user sees the common interests he has with someone else and he decides to *like* this person because of that. This like is shown in another person's SoInx event feed along with the common interests. *Like* does not reveal identity to the other user.

**Actors**

User

Peers (passive)

**Scenario**

User has opened SoInx and is looking at the radar and the different users show on it. One of the users displays a very interesting common interest and user has decided to like this person to show his interest. User clicks on this person and *view user* view is shown. User clicks the *like* button to send a like, to the other user.

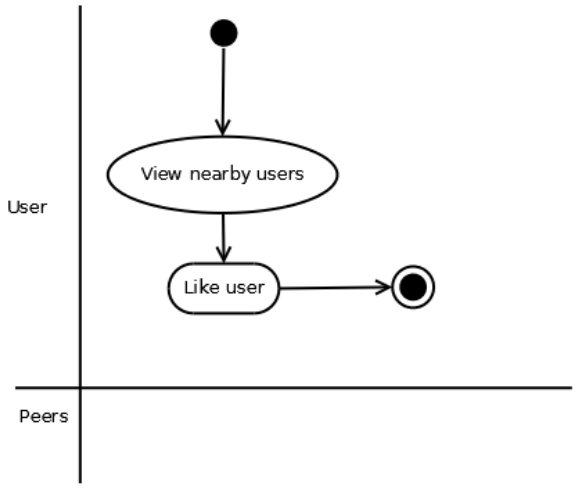


Figure 5.9: Activity diagram for liking a person.

## 5.9 View received likes and personal information

**Rationale**

The user can see if other users liked him or sent personal information.

**Description**

The user wishes to see if other users have liked him or sent him personal information.

**Actors**

User

Peers (active)

**Scenario**

The user has not checked the SoInx application for a while and wished to see has he received any *contact cards* or *likes*. He opens up the menu and chooses *event feed view*. *Event feed view* is opened by the UI and possible contact card and likes are fetched from memory. User can choose to view anyone of these by clicking it. User chooses to view one of the contact cards he received, this opens up the *received contact card view*.

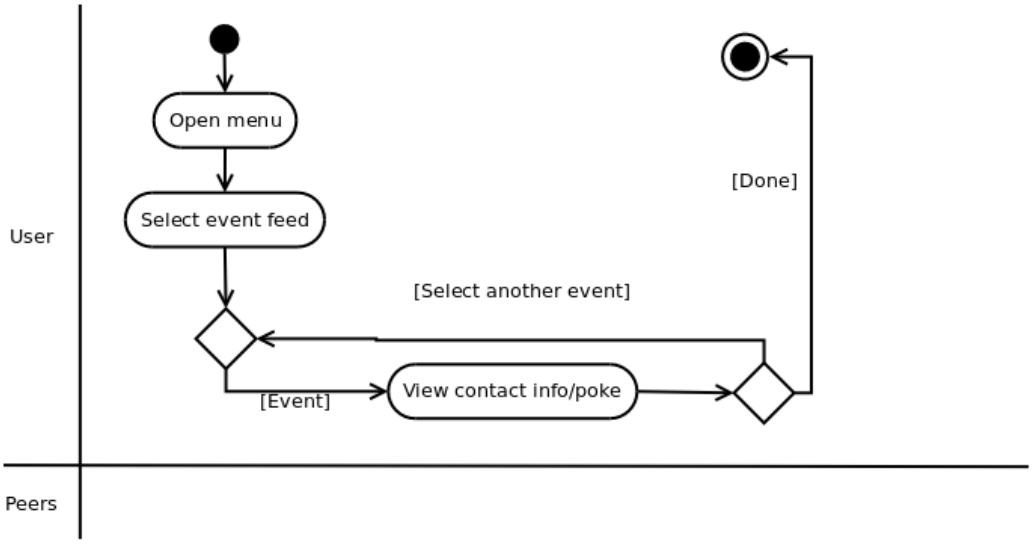


Figure 5.10: Activity diagram for checking received likes and contact cards.

# 6. ARCHITECTURE

SoInx was divided into two parts: the core and the UI. This division allowed for SoInx to be run without the UI during simulations. Within the core itself, inputs(the sources for interests) were created with a plug-in architecture in mind. This allows new inputs to be created for SoInx without making any changes to core SoInx.

Figure 6.1 shows the activity diagram for SoInx and how SoInx works. When another user is found in the network an evaluation algorithm is used to calculate social index for the content and to determine how important this is to the user. This evaluation algorithm is in the core, and it gives alerts to user interface when new interesting people appear. During simulations in place of user interface we can also utilize user behavioral model.

Depending on the action that user does on the user interface, or how behavioral model reacts, this feedback is returned back to the core to learning algorithm. This learning algorithm evaluates feedback and rates interest social indexes based on it. This information is passed to interest database and it helps evaluation algorithm to further decide which pieces of content are relevant and which are not. Content found in the network is also passed to the learning algorithm in addition to passing it onto the evaluation algorithm. The learning algorithm evaluates from how common or rare interest is and uses this to further calculate the index for an interest. For example Facebook like Tampere might be common in Tampere region and as such it would be rated lower than someone liking Turku in Tampere region.

## 6.1 Inputs as plugins

SoInx core was designed to include inputs as plugins, facilitating easy addition of new potential inputs. This was a major requirement since new social media websites come and old ones die away, allowing SoInx to stay on top of current trends. Plug-in design is implemented through the use of abstract interest and input classes. Creating new inputs is done by inheriting abstract input class and interest class for the new interests and interest source. These are then customized to be input specific, for example: New social media web site would require code for user authentication and data importation.

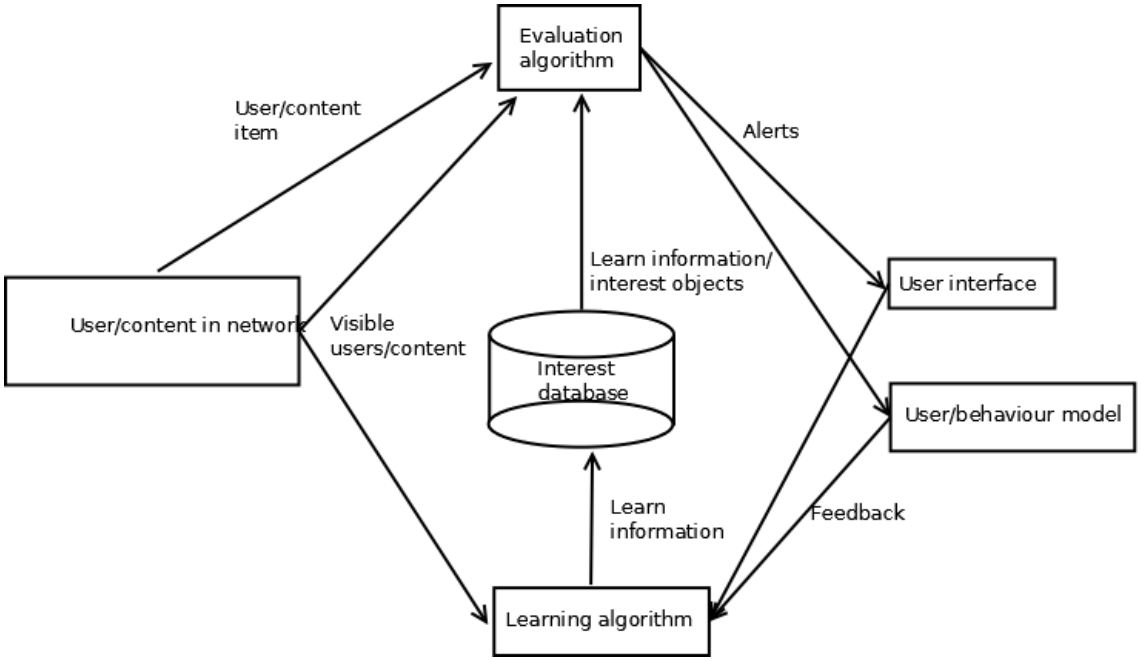


Figure 6.1: Activity diagram for SoInx.

## 6.2 Core and UI

Core and UI are separated in SoInx for three distinct reasons, memory and CPU savings, different UIs for different platforms and simulations. By allowing SoInx to work just the core, with UI shutoff, it saves a lot of memory and CPU processing power. This is especially important when simulating hundreds of SoInx applications in the simulator. As mentioned, with UI separated, we can construct different UIs for other platforms, with core staying the same. The third reason, simulation, meant that instead of UI we could use a simulation model that attempts to model user behavior.

In user tests, user actions had to be logged, such as clicking like or see more. But these clicks alone do not possess any useful information so we have to pair them with simulation time and other information, such as simulator user id. This meant that some of separation between UI and core had to be broken in order to facilitate retrieval of information from the core to the UI when logging, since logging is present in the UI.

Figure 6.2 displays the software architecture. In the figure social index engine is the same as the core mentioned previously. Social index engine was created with Python and Python libraries bencode, inotify, Kaa Metadata, PyPDF, Qt network and with local database. Social Index UI was created with PyQt. These tools are touched more upon in User tests section.

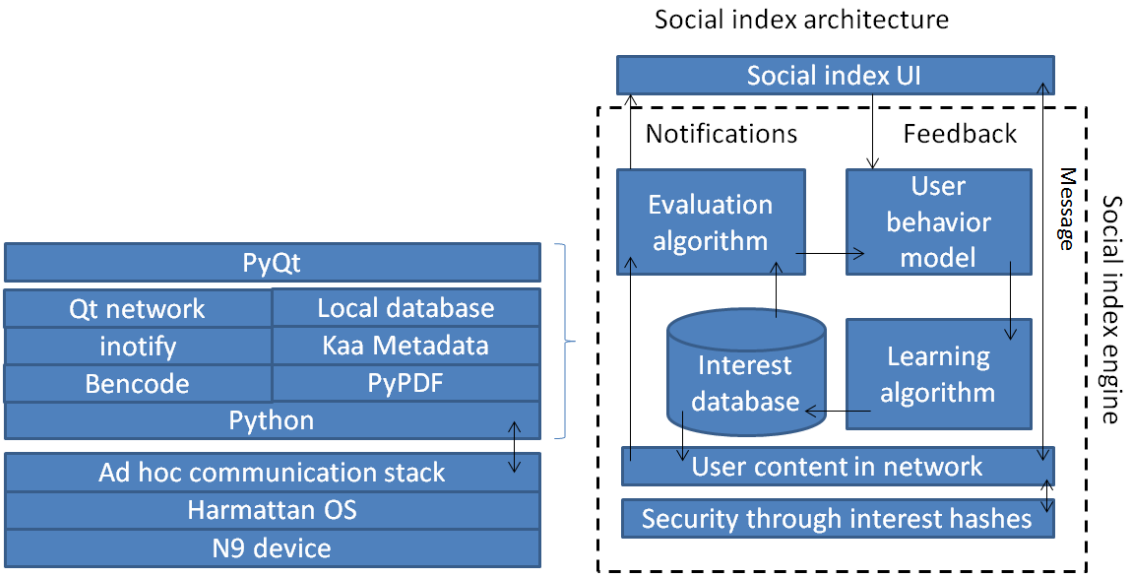


Figure 6.2: SoInx software architecture.

## 6.3 Maintaining privacy

Since privacy was an important issue for SoInx, cryptology method was needed to facilitate secured communication between users without revealing things that user did not want to reveal or the other user did not know. Shared secret was this method. Shared secret is based on both parties knowing something, usually a password [23]. In SoInx it was decided to use the interest common between users as a shared secret.

Each interest would have an identifier in human readable format that would be combined with salt, which was regenerated hourly. Salt in cryptography is random data that is used as additional input into a function that hashes information. In SoInx this salt is generated using device clock. This combination of identifier and salt would the be hashed with a hash function. These hashes cannot be decrypted without knowing both the salt and the interests. Any users that share these interests and have their device clocks synchronized can exchange these hashes without third party founding out what they hold. Once enough hashes have been exchanged we can form a secure communication link for chatting or other forms of communication.

Since salt changes hourly, that means hashes must be recalculated and stored again. Once they have been calculated, hashes can be compared to those found in the network. Changing hashes also make it very difficult to track people based on their hashes, this tracking data is valid only for an hour, after that hashes change.

# 7. SOINX IMPLEMENTATION

This chapter presents the various methods and tools used to develop SoInx. Methods were chosen because of their familiarity to research team and also because developing prototypes with them is easier than with other more common methods used in industry. Tools were chosen also for similar reasons, they were familiar to developers and prototyping with them is fast. Any time saved from learning new tools was time used on development.

## 7.1 Methods

Methods used while developing SoInx are same as those used in the development of Twin [24], since two of the original Twin developers were involved in SoInx and they had been proven to be excellent prototyping methods. These methods were fast iteration in development, immediate testing when feature had been developed and utilizing commit tags when making a commit to a Git repository.

### 7.1.1 Fast iterations

Fast iteration is process in which each developer designs and implemented a feature of the SoInx. Each developer is responsible for implementing and testing this feature before pushing it onto the Git repository. This was agreed upon methodology for developing since it had worked very well with previous prototyping project, Twin.

### 7.1.2 Immediate testing

It was agreed upon beginning of the development that before making any commits, each new feature should be tested. While commits to local branch may be done for better a documentation, pushing up to the main branch should not be done until feature was ready and working.

If developer found a bug, it was agreed upon that he would inform about the bug to person responsible for the original code. If developer had time for it and the person

Table 7.1: Commit tags used to document commit messages.

| Commit type | Description | # commits | % commits |
|---|---|---|---|
| Perfective | Adding or improving functionality | 386 | 45.36% |
| Corrective | Fixing incorrect functionality | 196 | 23.03% |
| Cleanup | Improving quality of code without affecting functionality | 69 | 8.11% |
| Documentation | Only adding missing comments, no changes to functionality | 21 | 2.47% |
| Workaround | Temporary or low-quality solution to a problem due to problems presented from elsewhere in the code or 3rd party libraries | 1 | 0.12% |
| Adaptive | Adding support for different target platforms | 15 | 1.76% |
| Preventive | Preventive incorrect functionality | 16 | 1.88% |
| No tag | Commit that for some reason did not possess tag | 147 | 17.27% |

who was responsible for the code was not present, he could attempt to debug it by himself. Thus code in SoInx was not owned by the developer who created it.

### 7.1.3 Tagging commits

When making commits to repository, commit was given a tag that described its intended effect. In SoInx there are six tags used: Perfective, Corrective, Cleanup, Documentation, Workaround and Adaptive. These tags were used in the previous twin project and as such documentation practices were derived from there [24]. Usage of these tags is explained in table 7.1.

## 7.2 Tools

Number of tools were used while developing SoInx prototype. Most of these were available immediately for use while others were made during development. Few custom tools were made during development using Python [25] or bash script [26]. Some tools were created for the purpose of documentation or for the user tests. Others were created to help with development.

Remove-unused-imports, upload and configure, mp3trunc scripts were used during the development to assist with cleaning or testing. Log _parser, and grep_commit_tags scripts were used with documentation or user tests. Table 7.2 shows all the utility tools used in developing SoInx.

Table 7.2: Utility tools used during development.

| utility tool | Description | Language |
|---|---|---|
| Remove-unused-imports | Removed unused imports from files | sh |
| grep_commit_tags | This would grep commit tags and count how many times each tag was used. | sh |
| log_parser | This script would parse various data from the user test log files | Python |
| upload | This script would upload SoInx source code to the N9 device | sh |
| configure | This was used to make sure all the required libraries were present in the system sh | |
| mp3trunc | This was used to remove all but meta data from mp3 files | sh |

### 7.2.1 Git

The version control system used in developing SoInx was Git. Git was chosen as the version control system for SoInx project due to its familiarity to developers and lack of central repository because of its distributed nature.

Distributed nature of Git nature means that there is not a central master repository where everyone must commit to, instead everyone has their own repositories which have their own branches of the source tree and they hold full version history.

Branch tree is a copy of the source tree in some way, at least in name. Branching a source tree allows each user to develop their own branch in parallel to the source tree. These changes in different branches can be merged together, forming a single source tree or simply to keep branches synchronized.

Because of the distributed nature of Git, each developer is able to independently develop project. They do not have to fetch changes made by others before committing their own changes. Each developer thus can develop new features independently of each other. Once feature has been completed and tested, it can be merged by the other developers [27].

### 7.2.2 Python

Python was chosen as the implementation language due to its familiarity to developers and because prototyping with Python is fast. Python is interpreted language with dynamic typing.

Whereas in C++ variable types are checked during compile time, this does not happen

in Python. Instead variable types are checked during runtime. Because of this, Python does not force variables to be certain types before they can be used.

Python includes extensive standard libraries and multiple third party modules for different tasks. For example there are two different python bindings for Qt. Syntax of Python is clear and very readable. Standard libraries in Python include modules to for interacting with the OS, file system, network and many more. It also includes basic data structures such as lists and has tables, called key-value dictionary in Python [25].

In addition to using standard libraries, several third part Python libraries were used in development: PyQt [8], Kaa metadata [28], pyPdf [29], Python-inotify [30], and bencode [31]. PyQt is discussed more extensively in the next section. Kaa metadata was used to access media metadata information, media here being audio or video files. Kaa metadata supports number of media formats, such as mp3 and avi. PyPdf provided access to PDF file metadata, title, author etc. Python-inotify provided access to feature in Linux which alerts when file or folder is accessed, this was used with media and PDF inputs to allow for learning from how often certain files are opened.

Bencode was used to serialize messages to and from simulator to SoInx [31]. Simulator facilitates communication between different SoInx applications but it also accepts different commands through these messages. These commands were used during user tests to slow down the simulation when something interesting was happening. SoInx also used bencode to serialize any messages it sends to the network.

### 7.2.3 PyQt

Qt is an open source library for the purpose of making graphical user interfaces. It was originally created by Trolltech, a software company based in Norway. It was bought by Nokia in 2006 and has since then been made into open source. Most recently it was sold to Digia. It is available for all popular desktop OS's and some mobile OS's. Qt is available under GNU GPL 2.1 for open source version and commercial version is under Qt Commercial License [32].

Qt was chosen from the beginning as the UI implementation language since it was known that the target platform would be a Nokia device. Originally PySide, bindings for Qt made by Qt organization was chosen but due its lack of maturity and other problems, PyQt was chosen instead. API in both PySide and PyQt are almost the same, difference is in importing libraries. PyQt is not available in Harmattan package manager as such we had to compile it on our own to N9 [7] [8]. This caused

number of problems and meant that we had to develop workarounds for several issues created by PyQt not working correctly in N9.

In addition to standard PyQt libraries, qtmobility libraries were also utilized in constructing phonebook input. Qtmobility library provides interface for accessing hardware and software features in the phone, such as GPS, camera or address book [33].

### 7.2.4 Platform

Nokia N9 was chosen as the platform because it was one of the few mobile platforms at the time which supported Python. Also part of the reasoning was that since previous projects had been made for N900, predecessor of N9, it would also be more familiar than completely new system. Final consideration point was that OS of N9 is a Linux distribution. Because it is Linux distribution, it made the available number of 3rd party libraries much larger.

### 7.2.5 SoInx Simulator

SoInx simulator is a network simulator that can connect thousands of mobile applications, called nodes, and simulate ad hoc networks for them. SoInx simulator simulates people moving around in a map. In this simulation they go to work, school, home etc. depending on the time of the day. Simulator facilitates ad-network communication to those nodes when they are near each other. SoInx simulator can be configured for different radio communication types with parameters. Default parameters for ad hoc WLAN were used during the user tests [2].

SoInx simulator uses a mobility model for constructing a model of movement of people throughout the map. This is generated through separate tool called SoInx mobility generator. SoInx mobility model is discussed more extensively in an article that was submitted for publishing [34] and in the thesis of Janne Kulmala [2].

# 8. USER INTERFACE

When opening different views, they stack on top of each other. The previous view being beneath the one most recently opened by the user. Closing a view and returning to previous one is done through the use of arrow button in the toolbar. In the main view arrow button is disabled and it does not do anything. Figure 8.1 displays a sequence diagram of moving between views in SoInx UI.

## 8.1 Main View

The main view in SoInx is called the radar. The radar displays other users nearby and how interesting they are to the user. It also allows users to access these other users and the common interests they share. Example of the radar view is shown in Figure 8.2. Center of the radar with the SoInx icon represents the user. Rings around it denote distance from the user. Each ring being equal to one hop. Hops in this case represent through how many devices packets must go before reaching user.

How much user has in common with another user and most common interest is displayed graphically in the icon representing another user. Icon background color gradually changes from light green to red depending on the index, index being average index of all the common interests. Icon size is 64 by 64 pixels, and it was

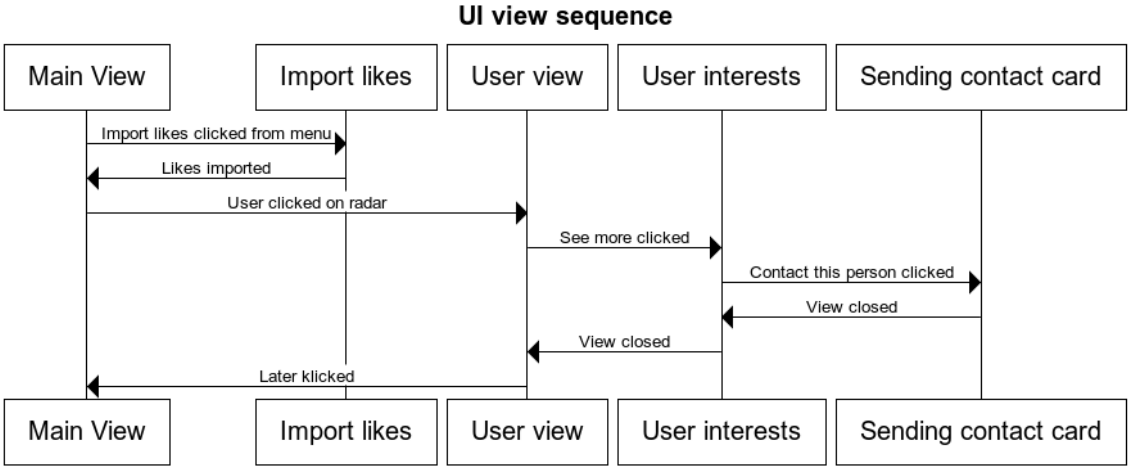


Figure 8.1: Sequence diagram of moving between SoInx UI views.


Figure 8.2: Screenshot of the main view, radar.

varied by 10 pixels, smaller being less interesting and larger being more interesting. Most common interests is denoted through the use of an icon common to the interest type.

Each represented user has an icon inside it that represented top interests. Audio type interest has a musical note as icon, video or audio type has a play button. Social media, like Facebook, has an icon of two persons while web type interests, such as URL's, have a globe. PDF files have an icon with PDF symbol. Also real persons, such as ones acquired from phonebook, have an icon with two persons. Once user has checked out another user, background color changes to the image of the most common interests. If most common interests does not have an image, it will simply change to black.

## 8.2 Import likes

The import likes view is used for controlling various sources for interests, inputs. Each of these inputs has its own mechanism for creating interests. Some of these inputs require their own UI for doing this. Example of the *import likes view* is shown in Figure 8.3.

Input is selected by clicking on it, and then click on the *Refresh* button. Depending on the input and its state, it will either open a UI or simply do interests refresh on

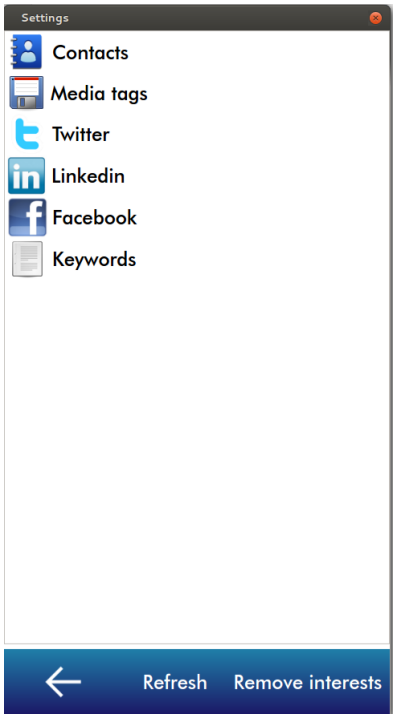


Figure 8.3: Screenshot of the import likes screen, where interest sources are added.

the background. Only Contacts input does not require UI, it fetches automatically contacts from phonebook and makes them into interests. Social media sites, Facebook, Twitter, LinkedIn, require log in. Refreshing them for the first time opens a web browser inside SoInx, using this browser user can log onto the site. Logging in gives SoInx an access token from the site. As long as this access token is valid, refreshing interests for this input does not open the web browser. Instead refreshing is done in the background.

## 8.3 Contact card

The *contact card view* was created for storing personal information of the user in case he wants to share personal information with another user. Example of the Contact card view can be found in Figure 8.4

The *contact card view* allows user to input name, nick name, email address, phone number, web address and personal message. Once user has finished with editing, he could save this information on the phone by clicking the *Save Contact information* button.

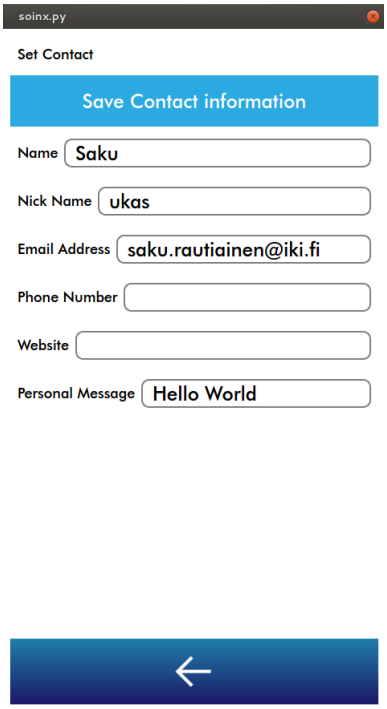


Figure 8.4: Screenshot of the Contact Card edit view.

## 8.4 User

The user view is shown when user clicks another user in the radar view. This opens up a menu which displays the same things as did the icon in the radar view, how interesting another user is and the most common interest. In addition there are several buttons for teaching SoInx. Example of User view is shown in figure 8.5.

The user view is divided into three distinct areas, top rectangle, middle rectangle and the four buttons. Top rectangle displays text that proclaims what type of interests you both have. Icon on the right displays the same information as did the icons in the radar view when another user had not been clicked. The rectangle below it displays on the right the image for the interest, if interest does not have image it then displays the same icon as above. Text on the left displays the friendly name for the interests.

The four buttons in the view are *Like*, *See More*, *Later* and *Not Interesting*. *Like* simply teaches SoInx that the user liked most common interests, it increases the index for the most common interest. It also colors the button and sends a like. This like is received by the other user and it is a signal that user liked the common things he has with the user. This is displayed in Event Feed.

*See More* button open the User Interests view which displays all the common interests. This also teaches SoInx, since user is interested enough to see the rest.

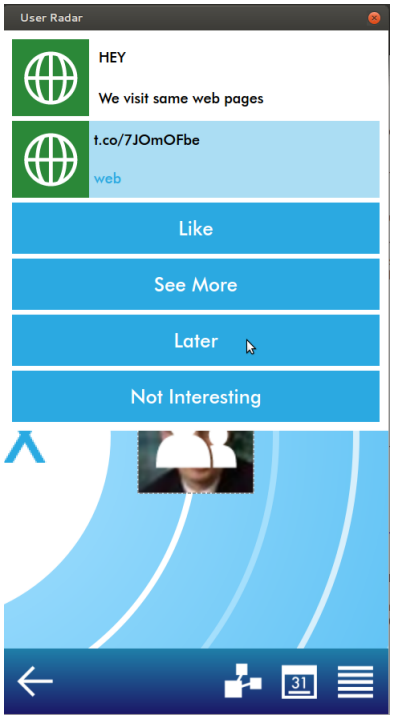


Figure 8.5: Screenshot of the user view.

*Later* button closes the view without teaching SoInx anything, it is a very neutral action. Whereas *Not Interesting* also closes the view but it also teaches SoInx, this time in a negative manner. It decreases the index of the most common interest.

Opening *user view* is animated, user view drops from the top of the screen. At the same time radar in the background zooms on the user which was clicked. When closing user view radar zooms out coming into full view and user view back up.

## 8.5 User interests

The user Interests view shows the common interests between user and another user. It also displays how interesting each interests is with the same color coding as in radar. Example of the user interests view is shown in Figure 8.6.

Top rectangle in the view is the same as in user View. Below it is the *Contact This Person* button and list of interests. *Contact This person* button allows user to share his personal information with the another user. In interest list each interests is shown in a rectangle shaped widget, image on the right is the interests image or icon based on the interest type and index if it does not have an image. Left to it is displayed the friendly name and below it is interest type.

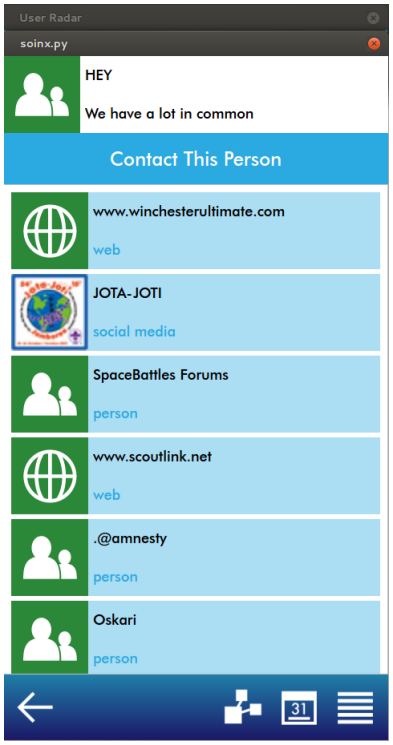


Figure 8.6: Screenshot of interests displayed in a list.

### 8.5.1 Sending contact card

The sending contact card view opens up when the user clicks the *Contact This Person* button in the user interests view. This view is used for selecting which personal information to share. Example of the sending contact card view is in the Figure 8.7.

The *sending contact card view* has two buttons in the top and six labels and six check boxes for personal information below them. *Chat* button is a place holder for chat functionality which was never implemented. *Send Contact Card* button sends the personal information user selected by using the checkboxes.

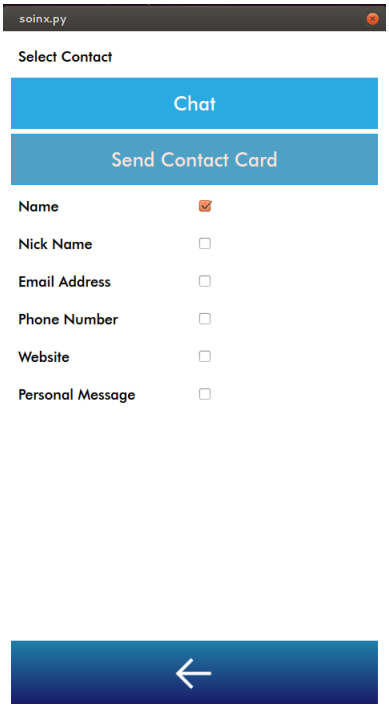


Figure 8.7: Screenshot of the sending contact view

## 8.6 Event feed

The *event feed view* opens by pushing the calendar button in the toolbar of the radar. The event feed displays likes and contact cards that have been received from other users. Example of the *event feed view* is in the Figure 8.8.

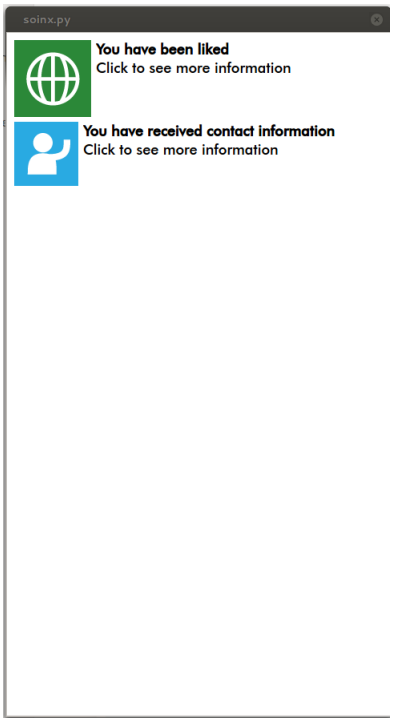


Figure 8.8: Screenshot of having received the like

### 8.6.1 Received contact card

The *received contact card* view displays the contact card another user has sent the user. This contact card displays only the information another user has sent. Example of the *received contact card view* is in the Figure 8.9.

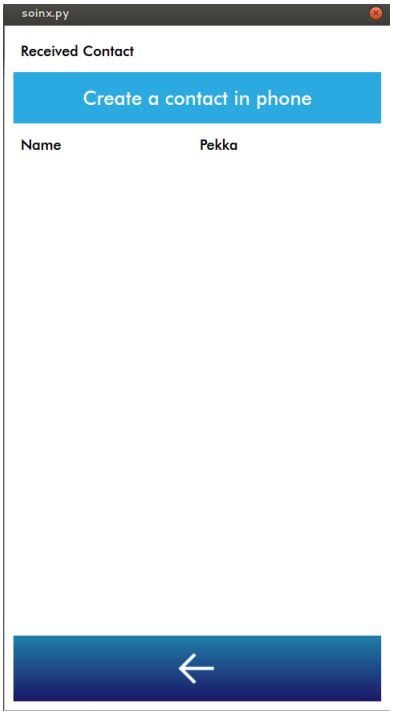


Figure 8.9: Screenshot of Contact info which was received.

## 8.7 Summary of the prototype interfaces

In contrast to Twin project, only about 24% of the code written for SoInx is UI code, as show in Table 8.1. This can be in part explained by the fact that the use cases in SoInx were not as complicated or feature rich since projects aim and this thesis aim is to create a prototype for new type of social interaction through finding information about people close by.

This chapter was used to display the user interfaces which implemented the use cases presented in Chapter SoInx use cases. In addition to that, there are other pieces of UI code that implement smaller functionalities or serve as abstract classes for several user interfaces.

Table 8.1: User view summary.

| View name | Module | # LOC | % LOC | Description |
|---|---|---|---|---|
| Main view | radar | # 289 | 5.18% | ... |
| Import likes view | settings_gui | # 95 | 1.70% | ... |
| Contact card view | contact_gui | # 66 | 1.18% | ... |
| User view | user_gui | # 111 | 1.99% | ... |
| User interests view | user_gui | 146 | 2.62% | |
| Sending contact card view | contact_gui | 62 | 1.11% | |
| Event feed view | events_gui | 134 | 2.40% | |
| Received contact card view | contact_gui | 62 | 1.11% | |
| Additional UI code | utils_gui, contacts_gui, style.css, qgraphicswebview.css | 379 | 6.80% | |
| total UI code | | 1344 | 24.11% | |
| Total lines of code | | 5574 | 100% | |

# 9. TESTING

A number of methods were used by developers to help testing SoInx. Development of SoInx was begun on PC with the intent that it could be simulated on a server, to enable further testing. Number of assertions were placed in SoInx core for assumptions, especially when making virtual functions since Python does not provide a mechanism for making pure virtual functions. Last we used linting software to find bugs in program through static testing.

## 9.1 PC version

Development of SoInx was begun on PC due to lack of modern mobile device. In the beginning mobile testing was done on N900, until the arrival of N9 mobile phones. Major reason for making PC version first and then importing it to mobile platform was because of the SoInx simulator[2]. This simulator allowed us to connect hundreds of SoInx softwares to simulator and through that simulate as if they were users moving about in a map.

## 9.2 Assertions

Assertions were used in the code to ensure that parts of the program work like they are supposed to. Assertion causes assertion error when run if the expression inside an assertion is not true. If assertion fails, program terminates and it reports where the assertion failed.

Since Python itself does not support virtual functions, assertions were put into abstract classes with empty functions to ensure when inheriting these abstract classes, those functions were created. This way we could have virtual functions without needlessly impairing security or testing of SoInx since assertions ensured we knew where the functions had not been implemented.

## 9.3 Linting

Linting is a form of static testing. Software that implement lint like features go through the code and flag possible errors, such as wrongly typed variables or syntax error compiler might not catch. We utilized pylint to find such errors [35].

## 9.4 Problems

Some of the problems with testing SoInx come from Python and the use of graphical user interface. Since Python interpreted language, most bugs are found when running the software, which makes finding bugs hard and time consuming. Some of these error could be found through static testing, aka code review and pylint.

Problem with graphical user interface is that there are not any good debugging or testing software to test UI. Testing has to be done by hand, which is also time consuming. Also UI had to be usable on N9, so we had to upload code for SoInx into the mobile device and this was time consuming.

# 10. USER TESTS

During the course of the SoInx project, ten user tests were conducted. These user tests were conducted to find bugs in SoInx and improve upon it if necessary. Additional motivation was to find out if users felt that application like SoInx would be useful and what kind of features should it have. Figure 10.1 shows user test in progress.

## 10.1 Methods

User tests were implemented using SoInx simulator and desktop version of the SoInx. Users would run through simulated 5 days with 900 users in area the size of Tampere. This would take about two hours and after simulation they would answer questions regarding the test, user experience and privacy concerns. Simulation was improved throughout the user tests, questions were added and made more specific. Randomization was locked after few tests. Users had the option of liking person, or saying they wished to chat or share personal information in the SoInx application.

Figure 10.1: User test arrangement. User tests SoInx on the simulator machine with a simulator map showing where he or she is while SoInx UI is next to the map.

Table 10.1: User tests logging results.

| User | # of meetings | # of meaningful meetings | # of unique likes | # of chat requests | # of contact sharing actions |
|---|---|---|---|---|---|
| User7 H | 343 | 55 | 10 | 0 | 0 |
| User6 H | 282 | 101 | 39 | 0 | 4 |
| User2 H | 360 | 79 | 34 | 1 | 1 |
| User3 H | 361 | 103 | 35 | 30 | 26 |
| User5 H | 361 | 122 | 61 | 18 | 30 |

Test data was acquired by crawling Facebook, by choosing users who had made their profiles public had their friends and likes downloaded. Since there are lot of users with public profiles in Facebook, we chose those had some association with Tampere, by using keywords associated with Tampere. These keywords included: Tampere, Näsi, Manse, Mustamakkara etc. After all the profiles had been downloaded they were first anonymized and then converted to SoInx profile format. Anonymization was done because what was important for testing was connections between people and what they liked. Anonymization also provided privacy to all those users on Facebook. These profiles were used to simulate other people in SoInx simulator.

Table 10.1 holds data from those user tests that logged various user actions. Halfway through the tests, test questions were altered and more logging was inserted, which is why only half of the testers are present in this table. Meetings describe the number of people user met during the simulation. Meaningful meetings are meetings where user had something in common with the other simulated user. Likes tells us how many of those unique meeting resulted in user liking other people. Chat requests and contact sharing actions describe similar actions that resulted from meaningful meetings.

## 10.2 User test analysis

Purpose of the user tests were to find bugs in the SoInx programming, as well as to determine if SoInx is something users would like and use. Majority of the users according to the final questionnaire were interested in SoInx, though some expressed the need for more features to make it more interesting. Seven users found finding new friends especially interesting, though one user did not believe that meeting someone through SoInx might result in contacting them in real life. Most users were not worried about the privacy or security that SoInx offered to maintain anonymity, but three users though that worst thing that can happen with SoInx is releasing personal information. Two users thought that constant chat or contact requests

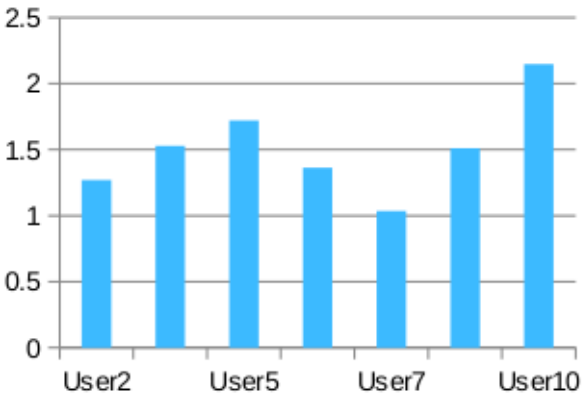


Figure 10.2: Graph of average number of interests per encounter for some of the users.

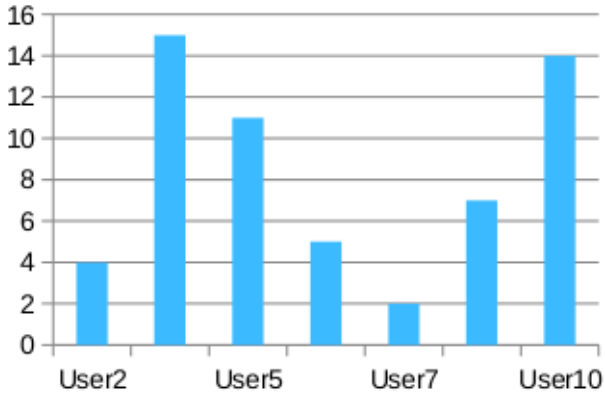


Figure 10.3: Graph of maximum number of interests per encounter for some of the users.

might be annoying.

Figure 10.2 displays the average number of shared interests between users. As seen from the figure, this average number is very small. Ranging from one to two shared interests. This implicates that either user did not have very much in common with anyone or with most people.

Figure 10.3 displays the maximum number of shared interests the user had with other user. While figure 10.2 showed that on average most people do not have a lot in common, figure 10.3 shows that there are cases where the number of shared interests is high. Singling out these few cases is especially important for SoInx to be a successfull product.

Some users thought that at times same interests kept popping up that they had already clicked as not interesting or that there were too many users shown on screen. This would be the result of SoInx not filtering them away. This could be improved upon by refining the learning and evaluation algoritms. More refined algoritms could filter away interestobjects that are very common or user has not shown interest in.

# 11. CONCLUSIONS

This thesis presented use cases, user interface and architecture to implement the use cases in a prototype social mobile application for automatic content discovery in ad hoc networks, SoInx. This thesis also includes presentation of the tools used to implement and evaluate the prototype application as well as the user tests conducted on the prototype application and their results.

Objectives of this project was to create a prototype application that can gleam data from mobile phones, transfer it over ad hoc network without compromising user privacy or security, while learning from data found in the mobile phone, network and from user interactions. Secondary objectives were to create an intuitive user interface that both attracts users and helps the program learn from user actions without being too annoying. First objectives were met with clear success, with the exception of learning which was developed until the end and due to lack of time was not perfected. Secondary objective of creating useful user interface was achieved with mixed results. While it supported learning from user actions, such as liking or chatting with other users, some user testers did not find it intuitive.

Methods for testing and evaluation were insufficient in part. While choosing Linux as mobile platform meant that application could also be run on desktop and thus speeding the development time, there were no automated tests for user interface. This meant finding bugs in the UI took a lot of time. User tests on the other hand showed that learning algorithm requires more refinement for SoInx to be a successful product.

# REFERENCES

[1] Toh, C.K. 2002. Ad Hoc Mobile Wireless Networks Protocols and Systems, 1st edition. Upper Saddle River, Prentice Hall Inc. 302 pages.

[2] Kulmala, Janne. DEVELOPING LOCAL SOCIAL APPLICATIONS ON MOBILE DEVICES. Thesis. Tampere University of Technology, 2013. Tampere: n.p., 2013. Print.

[3] Kulmala, J., Vataja, M., Rautiainen, S., Laukkarinen, T., & Hännikäinen, M. (2013). Social Index: A Content Discovery Application for Ad Hoc Communicating Smart Phones. Thinking About the Socio-technical Perspetives on Cloud Enabled Mobile Devices, 2013, 19.

[4] "Develop for the Nokia N9." Nokia Developer -. Nokia, n.d. Web. 14 Sept. 2013. <http://www.developer.nokia.com/Devices/MeeGo/>.

[5] "Maemo." Maemo. N.p., n.d. Web. 14 Sept. 2013. <http://maemo.org/>

[6] Paul, Ryan. "ArsTechnica." Ars Technica. N.p., 24 June 2011. Web. 14 Sept. 2013. <http://arstechnica.com/gadgets/2011/06/nokias-new-meego-based-n9-is-set-up-for-failure/>.

[7] "An Introduction to PySide." Category:LanguageBindings - PySide. N.p., n.d. Web. 14 Sept. 2013. <http://qt-project.org/wiki/PySide>.

[8] "What Is PyQt?" Riverbank. N.p., n.d. Web. 14 Sept. 2013. <http://www.riverbankcomputing.com/software/pyqt/intro>.

[9] "Nokia N9 UX Guidelines." Nokia N9 UX Guidelines. N.p., n.d. Web. 14 Sept. 2013. <http://harmattan-dev.nokia.com/docs/ux/>.

[10] "Getting Started Programming with QML." Qt Project. Digia, n.d. Web. 17 Oct. 2013. <http://qt-project.org/doc/qt-4.8/gettingstartedqml.html>

[11] Eagle, Nathan, and Alex Pentland. "Mobile Matchmaking: Proximity Sensing and Cuing." IEEE Pervasive Computing. To appear: April (2005).g

[12] Persson, Per, Jan Blom, and Younghee Jung. "Digidress: A field trial of an expressive social proximity application." UbiComp 2005: Ubiquitous Computing. Springer Berlin Heidelberg, 2005. 195-212.

[13] Nagowah, S.D., "Aiding Social Interaction via a Mobile Peer to Peer Network," Digital Society, 2010. ICDS '10. Fourth International Conference on , vol., no., pp.130,135, 10-16 Feb. 2010 doi: 10.1109/ICDS.2010.30

[14] Ben Dodson, Ian Vo, T.J. Purtell, Aemon Cannon, and Monica Lam. 2012. Musubi: disintermediated interactive social feeds for mobile devices. In Proceedings of the 21st international conference on World Wide Web (WWW '12). ACM, New York, NY, USA, 211-220. DOI=10.1145/2187836.2187866 http://doi.acm.org/10.1145/2187836.2187866

[15] Eric Paulos and Elizabeth Goodman. 2004. The familiar stranger: anxiety, comfort, and play in public places. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04). ACM, New York, NY, USA, 223-230. DOI=10.1145/985692.985721 http://doi.acm.org/10.1145/985692.985721

[16] Saarinen, Petri. USER EXPERIENCE OF PROXIMITY-BASED SOCIAL NETWORKING. Thesis. Tampere University of Technology, 2011. Tampere: n.p., 2011. Print.

[17] Author, Janne Kulmala & Author, Antti Laine & Author, Heikki Orsila, & Author, Jukka Suhonen, & Author, Marko Hännikäinen, 2013. Design for Device-to-Device Communication for Social Networking. Manuscript sumbitted for publication.

[18] "Symbian Platform." Nokia Developer -. Nokia Inc., n.d. Web. 29 Sept. 2013. <<http://developer.nokia.com/Devices/Symbian/>>.

[19] "Connect with Friends and Theworld around You on Facebook." Facebook. N.p., n.d. Web. 14 Sept. 2013. <http://www.facebook.com>.

[20] Twitter. N.p., n.d. Web. 14 Sept. 2013. <http://www.twitter.com>.

[21] LinkedIn. N.p., n.d. Web. 14 Sept. 2013. <http://www.linkedin.com>.

[22] "Packaging Your Application." Packaging Your Application. Nokia, n.d. Web. 14 Sept. 2013. <http://harmattan-dev.nokia.com/docs/library/html/guide/html/Developer_Library_Publishing_Packaging_your_application.html>.

[23] Menezes, A. J., Oorschot Paul C. Van, and Scott A. Vanstone. Handbook of Applied Cryptography. Boca Raton: CRC, 1997. Print.

[24] Laine, Antti. User Interface Prototypes for Social Ad Hoc Networking. Thesis. Tampere University of Technology, 2012. Tampere: n.p., 2012. Print.

[25] "Python Programming Language Official Website." Python Programming Language Official Website. N.p., n.d. Web. 14 Sept. 2013. <http://www.python.org/>.

[26] Newham, Cameron, and Bill Rosenblatt. Learning the Bash. 3rd ed. Gravenstein Highway North: O'Reilly, 2005. Print.

[27] "Git." Git. N.p., n.d. Web. 14 Sept. 2013. <http://git-scm.com/>.

[28] "Kaa.metadata." Kaa.metadata Kaa.metadata V0.7.5 Documentation. N.p., n.d. Web. 14 Sept. 2013. < http://doc.freevo.org/api/kaa/metadata/>.

[29] PyPDF. N.p., n.d. Web. 14 Sept. 2013. <http://pybrary.net/pyPdf/>.

[30] "Bitbucket." Bos / Python-inotify. N.p., n.d. Web. 14 Sept. 2013. <https://bitbucket.org/bos/python-inotify/>.

[31] "Bencode 1.0 : Python Package Index." Bencode 1.0 : Python Package Index. N.p., n.d. Web. 14 Sept. 2013. <https://pypi.python.org/pypi/bencode/1.0>.

[32] "Qt" Qt. Digia, n.d. Web. 14 Sept. 2013. <http://qt.digia.com/>.

[33] "Qt Mobility Project Reference Documentation." Qt Mobility 1.2:. N.p., n.d. Web. 14 Sept. 2013. <urlhttp://doc-snapshot.qt-project.org/qt-mobility/index.html>.

[34] Author, Janne Kulmala, & Author Antti Laine & Marko Hännikäinen, 2013 A mobility model for developing social device-to-device applications. Manuscript sumbitted for publication.

[35] "Pylint." (analyzes Python Source Code Looking for Bugs and Signs of Poor Quality) (Logilab.org). N.p., n.d. Web. 14 Sept. 2013. <http://www.logilab.org/project/pylint>.