UNIVERSITY OF TARTU
Faculty of Mathematics and Computer Science
Institute of Computer Science
Specialty of Information Technology

Sven Kirsimäe

F2F Mobile Computing

The Frid goes mobile

Master Thesis (20 CP)

Supervisor: Ulrich Norbisrath

Author: …….. "……" May 2009
Supervisor: ……..... "……" May 2009

Allow to Defense:
Professor ..... …… "……" May 2009

TARTU 2009

Contents

Acknowledgments

I would like to thank people who were beside me and helped in the process of writing this thesis.

At first I would like to thank my supervisor Ulrich Norbisrath. Under his guidance and encouragements throughout my master studies, the results of this thesis would never have seen the light.

I would like to thank my family, my mom Olga and dad Mati, who always understand and support me in everything I do. Special thanks goes to my sister Maria, who is far away but has never lost belief in me.

I would like to thank my coworkers, without who I would have never become professionally knowledgeable that was a prerequisite for this work. I'm very happy being worked with the best team ever.

I would like to thank all the friends who in the last minute reviewed this thesis - Tom Godber, Georg Singer and Jaanus Jaeger. Without your comments and reviews, the work would have never become as ripe as it is.

I would like to thank my friend, Priit Salumaa for taking over some of my other responsibilities at the time of writing this thesis. Without his help I could not allow myself the needed time and concentration for this work.

And last but not least, I thank a very special person who has brought light and warmth into my life and stood strong beside me in the hardest times of writing this thesis - thank you Irina.

Writing this thesis was a challenging task. Without a support from all of you I would have not enjoyed even half of it. Thank you all!

Introduction

In Spring 2007 I attended a course conducted by the Distributed Systems Group [27] at the University of Tartu [112]. The group's primary task was the establishment of the initial idea and implementation of a framework called Friend-to-Friend (F2F) Computing. It is a simple distributed computing concept where participants are each others friends and where computational tasks can be shared with each other as easily as friendship. The concept and initial prototype of F2F Computing was born. Since then, a substantial amount of related F2F work has been achieved [153]. During this time, I was mainly involved in mobile software development and did not further contribute to F2F Computing.

The main task of this thesis is a feasibility study on the use of F2F Computing in a mobile environment, alongside the creation of a F2F Mobile Grid infrastructure. It will describe the current state of the mobile industry and the constraints mobile software developers must work within. Implementation of the framework and demo application are provided as a practical part of the thesis.

Two decades after the emergence of the first mobile devices, mobile phones have evolved into more than just "walking-talking" devices. They come in many forms - different both internally and externally. Some are simple talking devices while others are communication tools possessing computation power comparable to that of a personal computer at the beginning of this century. The mobile phones are now such a natural part of our day-to-day lives that their presence is only noticeable when ringing is heard from someone's pocket. So essential is our primary need to communicate and be in touch with each other, that with billions of mobile subscriptions mobile phone have become more than a commodity device. In many ways they are also a statement symbolizing the owner. Presence, features and uniqueness of the mobile phone can enhance and define the life-style of the bearer. In its frenzy of features and requirements for personality and uniqueness, the mobile industry has become a fragmented, complex, constrained and in some cases industry of contradictions.

Mobile Grid computing is not a novel concept. Technologically it is similar to conventional Grid computing, except that the peers are highly mobile nodes. Whilst a single device is relatively limited, the total computational power of the mobile phones in a Grid has far greater potential. At first glance, creating a Mobile Grid, seems nothing more than an implementation issue. What may be initially overlooked is the inherent complexity and fragmentation of the mobile industry. Creating a successful Mobile Grid is more than a technological task, and is related to how we approach and use the mobile phones compared to the personal computer platforms that are running somewhere under our tables at our offices and homes.

This thesis represents a feasibility study for the use of F2F Computing in a mobile environment. Before beginning the implementation of a F2F Mobile Computing framework, deeper understanding of the mobile phone and mobile industry is required. This

thesis will describe the main constraints encountered in mobile software development and how these constraints can be overcome in order to port the current F2F Computing framework and ideology onto mobile devices. The practical part of the task involves building the F2F Mobile Computing framework. This proves the concept of a F2F running inside an environment that is highly mobile, generally very restricted and potentially limited on resources. Additionally, a practical demo application utilizing the implemented framework is created. Altogether, the work attempts to answer generic questions such as:

- Can mobile phones be used as a grid computing devices?
- What are the main obstacles in supporting grid-like frameworks on mobile devices?

The thesis is structured as follows. Chapter 1 will give a short history, status of the F2F Computing and define overall goals for the mobile implementation. Chapter 2 will give an overview of the mobile industry within the context of mobile software development. History of the industry, the status of the current market and the constraints of mobile devices are described. The chapter will end with ideas on how mobile grid-like applications could be implemented on a mobile platform. Chapter 3 will describe the work done in creating the F2F Mobile Computing framework for a selected mobile operating system. It includes the listing of tasks in order to port the F2F framework onto a mobile device. The chapter will be finalized with a summary of the achievements set forth in the beginning of the chapter. Chapter 4 will describe the demo application written for the F2F Mobile Computing framework. Documentation for the installation process, setup and execution of the application is provided. Chapter 5 will vision the future works for the F2F Mobile Computing framework that either were not a part of this thesis, were not achieved, are currently lacking or need improvement.

Chapter 1

F2F Computing

This short introductory chapter will give an overview of the work done described in this thesis. A short history and prerequisite status of the F2F project is given. The overall goals for the F2F Mobile Computing framework will be defined.

1.1 F2F is born

In Spring 2007 I attended a project conducted by the Distributed Systems Group [154][27] of the University of Tartu [112], where an idea of lightweight Peer-to-Peer (P2P) client boostraping was practiced and Friend-to-Friend (F2F) Computing was born:

> "Would it not be great, if you could just collect some friends in your widely used instant messenger client and then you could run your calculations on their computers and get the result much faster than making it only on your machine?" This was the vision of the project initiator Ulrich Norbisrath [153].

The main outcome of the project was to create a proof of concept technology where computing power could be shared as easily as possible between friendly peers that are known to each other. In the first version of the F2F implementation the setup and running of *ad hoc* F2F Computing was intended to be as "friendly" as possible. All you needed to have is the compatible instant messaging communication tool and friends online:

- The Peer-to-Peer (P2P) communication was implemented using out-of-the-box P2P communication technology provided by Skype [103]. The Skype4Java Application Programming Interface (API) [104] gives 3rd party applications (like the F2F project) access to Skype.
- SIP Communicator [102] was used as the multi-protocol (i.e. supporting MSN [71], Yahoo [124], ICQ [46], AOL [8] and Jabber [50]) customizable instant messaging (IM) tool to spread the computational tasks and coordinate the "computing friends" using a special F2F Computing plugin written for SIP Communicator.

The result of the project evolved into a new paradigm of distributed computing, merging ideas from peer-to-peer, High Performance Computing [44], and social networks (personal contacts, mutual trust and recommendation) in instant messaging. Work by [145] has structured and updated the initial prototype into more stable and usable

Java-based F2F Computing framework for general usage. The update did not deal with issues like security, heterogeneity, fault tolerance and other issues except the base functionality of the framework. The F2F prototype made in 2007 was updated so that simple distributed programs can be written using a pure object-oriented approach and widely available communication tools. The project was still undergoing daily development and active projects ([150, 144, 156]) at the time of writing this thesis. The official status of the project at the time of writing this thesis was [145, 153]:

- The Java-based F2F Computing framework enables:
  - forming of the F2F Grid (Frid) using instant messaging tools (like SIP Communicator [102]);
  - composing the Frid Jobs[1] using the predefined interface;
  - submitting Jobs to the formed Frid;
  - executing Jobs as Tasks[2] by the Peers[3];
  - inter-Tasks communication depending on the implementation and requirements of the composed Job;
  - gathering of the results from the Frid.
- F2F is written in the Java [55] language as it is cross-platform, easily portable [53] and had other features that are required by the F2F network (like custom class loading in order to execute foreign Java code sent as a Task).

1.2 The Frid goes mobile

Porting the F2F Computing framework to a mobile platform would prove the concept of F2F to be able to run inside a highly mobile, possibly very restricted and potentially limited resources environment. The main features of the F2F Mobile Computing framework are based on the inherited architectural goals of the existing work (see Section 1.1). The technological targets for the F2F Mobile Computing framework were defined to:

- enable the core of F2F Computing - composing Jobs and distributing Tasks for the mobile Peers to compute;
- be lightweight - easy setup, management, usage and extendability;
- enable use of IM communication tools for easy personal and social trust-based F2F communication;
- use P2P techniques fostering fast/direct communication between the Peers (like NAT traversal [148] and direct TCP-IP sockets) when the initial IM communication channel underperforms. The framework should always use the best available connection;

[1] A Job is a computational problem that can be solved in parallel manner.

[2] A Task is executing part of the Job than can be run independently.

[3] A Peer corresponds to the entity in control of one of the users (friends) participating in solving a respective Task.

- be supported on as many mobile devices as possible.

The successful porting of the F2F Computing framework is dependent on many factors. The porting itself is not expected to be a trivial task and changes to the existing work is anticipated. Before applying concrete changes (see Chapter 3), knowledge of mobile industry is required in order to understand what kind of problem domain is being dealt with (see Chapter 2).

Chapter 2

Review of the mobile industry in the context of software development

In this chapter the issues found in the mobile industry through the eyes of a mobile software developer will be introduced. The scope of this thesis only scrapes surface of the mobile industry, where hundreds of different players are present composing a very lively ecosystem [165].

A short history and current state will be presented to explain current constraints in the mobile hardware (see Section 2.2.1), software (see Section 2.2.2) and operator (see Section 2.2.4) domains. A more detailed overview of the Symbian operating system, as a potential F2F Computing framework development base, will also be given (see Section 2.2.3). This chapter will end with thoughts on how grid-like mobile applications would position themselves within the current state of the mobile industry (see Section 2.3).

2.1 Mobile devices

You can tell what a culture values by what it has in its bags and pockets. Keys, combs and money tell us that property, personal appearance or trade matter. In the seventeenth-century, the personal watch, being a rarity, was baroque high technology, a compact complex device that only the most skillful artisans could design and build. Their proud owners bought not only the ability to tell the time, but also bought into particular values: telling the time mattered to the entrepreneurs and factory owners who were busy. Pocket watches provide the closest historical parallel to the remarkable rise of the mobile cellular phone in our own times. Pocket watches, for example, started as expensive status symbols, but by the twentieth-century most people in the West possessed one. When cellular phones were first marketed they cost the equivalent of a small car - and you needed the car to transport them since they were so bulky [138].

A mobile phone or mobile (also called *cellphone* and *handphone*, as well as *cell phone*, *cellular phone*, *cell*, *wireless phone*, *cellular telephone*, *mobile telephone* or *cell telephone*) is a long-range, electronic device used for mobile voice or data communication over a network of specialized base stations known as cell sites [68]. In addition to the standard voice communication feature of a mobile phone, current mobile devices are able to support many other additional mobile-related services such as Short Message Service (SMS), Multimedia Messaging Service (MMS), email communications, Internet browsing, camera and video recording, music player, radio and Global Positioning

System (GPS) services. This list makes the device nowadays a rather complex piece of machinery, both in hardware and software (for more about mobile complexity see Section 2.2).

Present statistics show billions of subscriptions to the cellular phone services (for more see Section 2.1.3) making a mobile handset nothing more than a commodity. Like the pocket watch - a device carried everywhere, on the person, by everybody. Beyond that - a lifeless brick of technology is becoming alive, a status marker for the owner with its ability to be customized not only in its looks but also the software running on it (for more on social effects see Section 2.3).

2.1.1 Long live the mobile

Cell-based mobile networking [22], as we know it today, is relatively young technology. Cells for mobile phone base stations were invented in 1947 by Bell Labs [16] engineers at AT&T [15] and further developed by Bell Labs during the 1960s. Recognizable mobile phones with direct dialing have existed at least since the 1950s. But the technology required infrastructure that was missing for public usage in those times [68, 45]. In the late 1970s infrastructure started to emerge all around the world. The first commercial launch of cellular telecommunications was launched in Tokyo, Japan in 1979. In 1981 the cellular networks were launched in Denmark, Finland, Norway and Sweden. These networks were so called 1st generation analog telecommunication networks (1G). The first mobile devices for these networks were heavy and expensive. Motorola DynaTAC (Dynamic Adaptive Total Area Coverage) 8000X [61] was released in 1983, weighed 793 grams and was 25 cm high excluding the antenna which increased the length of the device by additional 2/3rds. The battery allowed a call of up to 60 minutes and recharging it took 10 hours. The device was considered as revolutionary at the time, because contemporary mobile devices still had to be installed into vehicles. The DynaTAC 8000X was the first true portable handset.

Second generation (2G) mobile networks started appearing in the 1990s. By 1995 most of Europe had 2G coverage. The main difference between 1G and 2G was the switch to digital speech signaling in the 2G networks. The most common 2G technology families is GSM (Global System for Mobile communications), which is still widely used today and is the basis for new mobile network technologies. 2G introduced data services for the mobile. The first manifestation of this was the appearance of Short Message Service (SMS). With the development of more capable mobile networks, the miniaturization of digital components and the development of more sophisticated batteries, mobile phones themselves have become smaller and lighter. This progress has made mobile devices more popular.

Mobile usage statistics also started appearing once 2G technologies were established. The International Telecommunication Union [63] has gathered global statistics about mobile cellular telephone subscriptions for more than a decade. In 1998 the penetration was about 6%-8% and has been steadily rising, reaching more than 60% in 2008. Over the same time span fixed telephone lines have stayed at the same penetration - increasing from about 14% in 1998 to about 18% in 2008. Mobile cellular telephone subscriptions have grown much faster than other communication technologies like fixed broadband Internet access. Mobile cellular subscriptions more than doubled between 2002 to 2007.

3rd generation (3G) mobile networks are now emerging. The first commercial 3G

network was launched in Japan, in 2001. From Asia, the technology started to appear in Europe, the United States (US) and other countries. The main feature of 3G is its increase in data transfer speeds[1]. Increasing data speeds have been correlated with increasing cellular data usage [62]. 3G networks are still being run alongside current 2G *de facto* networks. The move can be complex, as whole new mobile infrastructure has to be built, with care taken to not compromise the existing 2G networks. Netsize statistics [137] show around 30% of overall mobile subscribers (which is around 3 billion) using 3G networks in 2009, mainly in Japan where it is now the main network technology having superseded 2G. The US is second globally. In Europe, the leaders are Italy, the United Kingdom and Germany.

The move from 1G to 2G to 3G has not been as idealistic and linear as the summary suggests. The historical events described are merely major milestones marking the evolution of mobile communication technology. The scope of the current thesis does not cover the specific developments that occurred between these milestones, both in technology and politics. The mobile network operators themselves have become industrial giants and are separate entities from the mobile device manufacturers. The first are mobile communication service providers, whilst the second are end-user device providers for the existing, sometimes regionally different, mobile communication infrastructures owned by the operators. Both of them trying to please the ever demanding customer. Their dependent but competitive relationship, combined with the incredibly fast pace of innovation, has introduced fragmentation and complexity to the inherent constraints of mobile technology, where all parties involved need to be satisfied (issues on the mobile technology constraints will be described in Section 2.2).

Nowadays several categories of mobile phones exist, ranging from handsets with basic voice-only functions, through feature phones (often targeted at niche market segments like music, e-mail, GPS or photography), and on to smartphones which are in effect small computers The technology filled smartphone is not suitable for every user, and the device segmentation is a response to market pressure. Some customers really do require only a simple voice-communication device; others benefit from the convergence of features that remove the need to carry separate camera or music player devices; still others require a device that operates as an "ultimate" communication tool where SMS, e-mailing, World Wide Web browsing and other Internet features can be used as appropriate from anywhere. The more personal the device becomes, like the pocket watch, the more personalized its features are expected to be. The specialized requirements of each device type, and the commercial pressure to innovate ahead of competitors, has led to a situation where there are a hugely diverse range of hardware specifications and software platforms to work with. Technical leaps and changes in direction in the mobile device industry can occur within a handful of years, whilst legacy devices remain in use alongside the new devices (for more see Section 2.1.2).

2.1.2 Change in the wind

The mobile industry is not homogeneous for reasons of history, competitive pressure and consumer choice. It is horizontally segmented. It has many different players from operators, handset manufacturers to content providers. Each of them is in their own horizontal segment in the industry. The first two, operators and handset manufacturers, are the oldest. Content providers are becoming stronger and growing new markets, in

[1] data rates up to 14.4 Mbit/s on the downlink and 5.8 Mbit/s on the uplink

some cases taking some control and customer spending away from the elders. As a result, mobile software development is emerging as a separate industry. Operators and manufacturers provide the platform for this.

Events happened in recent years also show clearly that the industry is moving more into a vertically integrated structure. In mobile technology this means a full mobile platform for users and content providers intersecting the device, software (both internal and external) and the operator services. The vertically positioned mobile solutions include:

- software development kit (SDK),
- go-to-market route(s) (like application stores),
- industrial design (look and feel of the mobile device),
- user interface (UI) customization possibilities,
- core applications and services,
- operating system (OS) and/or application environment, and
- hardware platform (mobile phone and networks).

Good examples in this positioning are the new Apple iPhone [9] (release date June, 2007) and Google Android [7] (release date October, 2008) mobile devices. They are tightly integrated with the software platforms in and outside of the mobile device. Even more astonishing is the fact that these solutions are both recently released and already stand strongly against the well positioned mobile industry leaders like Nokia [72], Qualcomm [94], Microsoft Windows Mobile [65], Adobe [3], Intel [58] and LiMo [60] (for more detailed statistics see Section 2.1.3).

These vertical mobile platforms allow mini-ecosystems of content providers to evolve within them. Currently, the most successful example fostering this is the Apple Appstore [10] with more than 65,000,000 application downloads per month as of end of 2008 [166]. All this is possible because of the full vertical positioning of the Apple iPhone mobile platform[2]. The success of the Apple Appstore is being emulated by other mobile industry players like already existing Android Market [6] for Google Android mobile devices. Mobile software markets are now something that every major mobile industry leader is trying to achieve - Research In Motion's (RIM) [96] BlackBerry App World [17], Palm's application store [83], Nokia's Download Store [72] and Ovi [82]. Many markets are still to be opened and some of them will be managed by operators, like Orange's App Shop [80], Verizon's Appzone [117], Sprint's OnDemand [20] and T-Mobile's application store [21].

Mobile application aggregators (like GetJar [39]) have existed for a long time, but platform specific software markets are getting stronger and have already been proven to be successful when vertically positioned. Software for mobile devices is booming. In many cases it is becoming a commodity where the novelty[3] and differentiation[4] phases are already history. Mobile software has become an essential commodity for the hardware. Like software inside a car radio, there is a base level assumption of

[2]and additionally because of the more acceptable revenue model for the content providers.

[3]phase of the software being new and the demand is scarce.

[4]phase of the software being competitive with the other software products.

what software should be inside a mobile device. For example, nowadays it is expected that an e-mail client is pre-installed on most mobile devices. Emerging mobile software markets are contributing to the frenzy in the mobile software industry.

The recent changes requires for all the mobile industry players to work together and cooperate in order to succeed. Maybe one day all this will remove the current obstacles found in the mobile software development (see Section 2.2) and they will be nothing more than a history to remember.

2.1.3 Current status

As described in the Section 2.1.2 changes in the mobile industry may happen very fast. Getting valid data representing the current market is complex. Generally, it is based on the information that is provided by a party involved in the mobile industry. It can be a company with a large volume of mobile traffic going through, or a company having access to the statistics itself:

- operator - knows the mobile devices that are using its infrastructure;
- mobile handset manufacturer - knows how many mobile devices are shipped;
- a mobile software aggregator, a repository of mobile software for public use (GetJar [39], 148Apps [48]) - knows what kind of mobile software is being accessed;
- mobile advertisement provider (AdMob [1]) - knows where the ads are displayed;
- website tracker (StatCounter [106]) - knows what device is accessing a site it's tracking;
- private market analysis and consulting companies (Gartner [37], comScore [24], Netsize [137]) - know the business they analyze or consult.

In the scope of this thesis, I am interested in the current status of:

1. the mobile device manufacturers,
2. and the connected devices.

Both of them will hint at the possible candidates for the mobile phone platform suitable for the F2F Computing mobile framework. From the mobile device manufacturers I can narrow down to a more popular mobile operating system and the device; from online statistics I can see which mobile phones are used mostly for the networking tasks.

Reports from private consulting companies are the most complete but are generally expensive. In some cases, they may provide valid free materials either through the press releases or as full reports. For operators or handset manufacturers data about the mobile industry is concerned as their core business. Thus the data can be either classified or provided in a limited way. In the light of a rising mobile software industry some of the parties are providing free statistics that are relevant to their own business. The intersection of these materials gives an overview of the current status of the mobile industry.

GetJar is a mobile software application distributor, an aggregator founded in 2004. Beginning from the October 2008 GetJar averages over 14,000,000 downloads per month. At the time of writing this thesis the total amount of downloads is 300,000,000.

Rank	Mobile Device Manufacturer	Market Share
1.	Nokia	54.6%
2.	Sony-Ericsson	17.63%
3.	Samsung	4.95%
4.	Motorola	2.82%
5.	BlackBerry (RIM)	2.78%
6.	LG	1.29%
7.	Huawei	0.31%
8.	Siemens	0.21%
9.	HTC	0.17%
10.	Sagem	0.14%

Table 2.1: Worldwide manufacturer market share by GetJar (March, 2009).

The number of supported mobile devices is over 1300 and the total amount of downloadable content is more than 35,300 mobile applications [39]. GetJar provides free statistic information based on its traffic. Table 2.1 lists the mobile devices by manufacturers accessing the GetJar site with statistics-snapshot taken from March, 2009. It is very clear that Nokia and Sony-Ericsson [105] are currently leading mobile handset manufacturers.

AdMob, founded in 2006, is a mobile advertising marketplace for mobile web. AdMob provides monthly mobile metrics reports based on the ads from more than 6,000 mobile web sites and 1,000 applications from more than 160 countries. Since the launch, AdMob has served more than 50 billion ads. AdMob stores and analyzes the data from every ad request, impression, and click [136]. The February 2009 report listed the top smartphones, a mobile phone which has identifiable operating system (for more on mobile operating systems see 2.2.2).

Table 2.2 lists the most common mobile operating systems found by AdMob in February, 2009. Notably the Apple iPhone OS is not found from GetJar's statistics listed in 2.1. The reason for this is that GetJar, as any other mobile software aggregator, does not provide software for iPhone (see Section 2.1.2). High traffic found in AdMod is because the iPhone is very popular mobile Internet browsing device in the United States with around 50% of presence. Its popularity has risen around 30% within the last 6 months as in August 2009 iPhone was accounted for only around 4%.

In the context of connected devices, table 2.3 provides the similar data with GetJar. Again, it has to be noted, that AdMob's statistics is based on mobile Internet browsing whereas GetJar's data is mainly based on mobile software application downloads. Mobile Internet browsing requires more advanced mobile devices and higher mobile hardware constraints apply (see Section 2.2.1). This is the reason for the statistics being different.

StatCounter is a free web tracker [106]. It monitors the visitors on a website providing real-time statistics with detailed visitor tracking and analysis. From December 2008 StatCounter have started gathering mobile related web traffic. The source is to confirm our previous findings. In March 2009, Apple iPhone was found accessing the sites worldwide for around 32-42%. A close call to Symbian mobile operating system (OS) being counted for around 32-47%. It is clear that the iPhone, although being a relatively new mobile phone, has become a popular Internet browsing device. The main reasons for that are the loosened limitations in hardware and software constraints

Rank	Mobile Operating System	Feb 09
1.	Symbian	43%
2.	Apple iPhone OS	33%
3.	BlackBerry (RIM)	10%
4.	Microsoft Windows Mobile	7%

Table 2.2: Worldwide mobile operating systems by AdMob (February, 2009)

Rank	Mobile Device Manufacturer	Market Share
1.	Nokia	30,2%
2.	Apple (iPhone)	17,9%
3.	Samsung	9,8%
4.	Motorola	9,4%
5.	Sony-Ericsson	9,0%
6.	LG	3,6%
7.	BlackBerry (RIM)	3,3%
8.	Kyocera	2,1%
9.	HTC	1,6%
10.	Palm	1,1%

Table 2.3: Worldwide handset data by AdMob (February, 2009)

especially for World Wide Web access.

In general it is clear that the Symbian [109] mobile operating system still has a very strong ground in mobile OS market (for more see Section 2.2.2) and Nokia is used as a popular Internet browsing device making it also a capable device for networking tasks.

2.2 Constraints of a mobile device

In Section 2.1 I define a mobile phone: a mobile phone or mobile (also called *cellphone* and *handphone*, as well as *cell phone*, *cellular phone*, *cell*, *wireless phone*, *cellular telephone*, *mobile telephone* or *cell telephone*) is a long-range, electronic device used for mobile voice or data communication over a network of specialized base stations known as cell sites [68]. Having many names one can imagine that this device may come in many forms both in looks and works. Modern mobile devices are packed with different hardware and software where its primary function of voice communication can be a merely another feature. The project WURFL (Wireless Universal Resource File) manages a database that contains information about all known wireless devices on the Earth. Of course, new devices are created and released all the time. While the WURFL configuration file is bound to be out of date days after each update, chances are that the WURFL lists all of the wireless devices currently available [155]. The project is an open-source. It is intended and managed by developers working in the mobile industry. Simple search on the database reveals around 5,000 different mobile devices listed[5]. This is a listing of only mobile device models. It excludes the regional marketing related renaming and in some cases simple look-and-feel modifications on the same devices being released.

[5]with the current date of April, 2009.

All the mobile devices have some kind of input and output interface and other hardware features. In general the input interface is a some kind of keypad and buttons for controlling the device. The output interface generally is a screen for communicating with the user. Hardware features define the device's capabilities (for more on hardware constraints see Section 2.2.1). Additionally mobile devices run some kind of software. There are hundreds of vendors that have emerged in the last ten years offering embedded software like operating systems, multimedia & graphics engines, middleware and core applications, application environments, browsers, on-device portals and active idle screen solutions. The software itself can differ by version within the model as it depends on the firmware updates applied to the concrete device (for more on software constraints see Section 2.2.2). Operators will provide hardware and software requirements to the handset manufacturers as fixed, one-off requirements. This is a process manufacturers struggle with, given that operator specifications often exceed 4,000 requirements and are refreshed every six months [146]. Their main interest is to have mobile devices suitable for (only) their networks and introduce operator restrictions (for more see Section 2.2.4).

The problem described above is called *mobile fragmentation*. For example, in the domain of mobile software development, developer have to accept the existence of many different devices. Compared to the PC world, where developer have to generally accept some popular hardware platforms, operating systems and screen resolutions, this "some" is a multiple of many in the mobile industry. Mobile fragmentation introduces massive constraints to the third party mobile software as F2F Communication framework.

2.2.1 Hardware constraints

Mobile hardware constraints fall into three main categories:

1. input constraints,
2. output constraints, and
3. power constraints.

Input constraints

Input defines how a user can control the mobile device. Generally it is implemented by a set of buttons on the device which usually bear digits and other symbols but not a complete set of alphabetical letters. Mobile phones also have different additional buttons for specific tasks. It is an important fact, that mobile devices in general do not have the point-and-click interface of a normal PC. It is just too compact for this kind of interface.

Figure 2.1 layouts the generic layout of a common mobile keypad. It is divided into two separate areas - one for navigation and the other for dialing and character input. The navigation keys are *MOVE UP*, *MOVE DOWN*, *MOVE LEFT*, *MOVE RIGHT*, *SELECT*, *SELECT_1*, *SELECT_2*, *ACCEPT CALL* and *DECLINE CALL*. The latter two are self explanatory for responding on a phone call. Previous buttons are for navigating within the device software (see Section 2.2.2). These buttons are also sometimes referred as *5-way navigation keys* and *softkeys*. Phone number dialing and character input is done using the keys below the navigation keys. The layout and

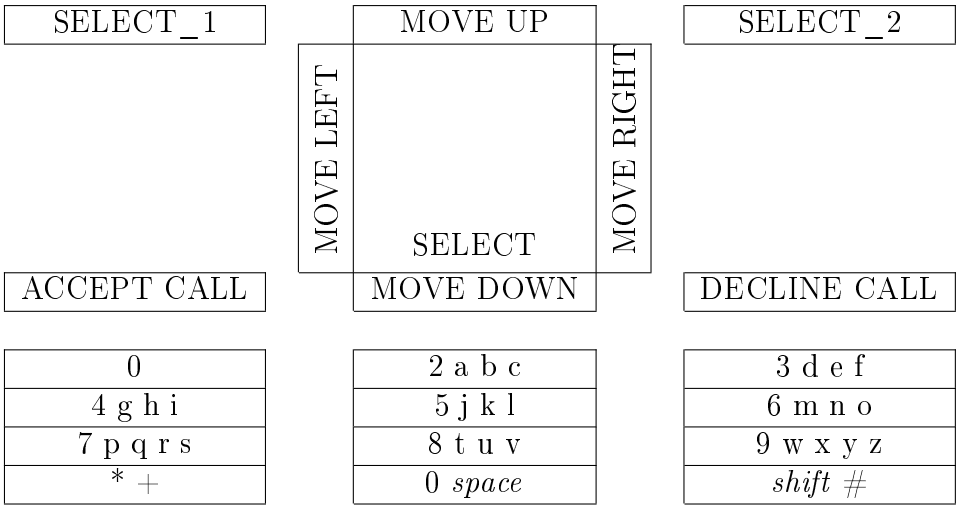

Figure 2.1: Generic keypad layout of a mobile device.

alphabet of the keys is historically from the telephone exchange names [114]. To enter a character (or a number) the button has to be pressed many times until it is its turn in the character sequence shown on the button (for example, for *r* press button *7* for three times in character mode of the keypad).

The main limitation of input comes from the size and non-standardized navigation and custom keys. Imagine typing a message using keys with the size of 0.8 x 0.8 centimeters in a subway speeding at 80 km/h (assuming the subway has mobile network coverage). The result can be full of typos and the overall experience may not be as easy as it seems. As non-voice communication is becoming more relevant using the mobile devices (see Section 2.1.3) so the different keypads have appeared to leverage the input constraints. Emerge of the *SureType* [108], full QWERTY keyboard alike keypads and *Trackballs* [116] are stirring more the current status of the input fragmentation.

Output constraints

Output defines how the user gets information back from the device. Over the years the mobile device screens have evolved from a very simple digits-only (the first Motorola DynaTAC 8000X mentioned in Section 2.1.1) through black and white low resolution screens to high resolution screens with millions of colors (see Table 3.2 in Section 3.2.5).

J2ME Polish [49] is a similar database to WURFL described in Section 2.2. It also profiles mobile devices and lists out their specifications. The difference is that it lists only JME-enabled devices (see Section 2.2.2). According to the current statistics (see Section 2.1.3) Nokia is one of the most popular mobile devices. Table 2.4 lists the different screen resolutions available by J2ME Polish at the time of the writing this thesis. The list does not include possible differences in the number of colors supported by the screen and announced devices released in the future like Nokia N97 with screen resolution of 360 x 640 pixels (expected June, 2009).

The main constraint of the output lies in the relatively low screen resolution and size to visually represent data in a meaningful way. Due to the fact that many screen resolutions need to be supported mobile software development can be a complex and resource intensive endeavour.

Screen resolution of width x height in pixels
96x65
128x128
128x160
176x208
208x208
240x320
320x240
352x416
640x200
640x320

Table 2.4: List of different JME-based Nokia mobile phone screen resolutions

Power consumer	Power consumption
Bluetooth	5%
Other	15%
Memory	20%
CPU	20%
Wi-Fi	40%

Table 2.5: Approximation of power consumption elements found in mobile phone

Power constraints

Power constraints of a mobile device fall into two categories:

1. Energy consumption of the mobile device.

2. Processor speed and memory limits of the mobile device.

Today's mobile devices come with many different wireless communication interfaces like Wi-Fi, Bluetooth, RFID (Radio-frequency identification), GPS (Global Positioning System) receivers, faster CPUs, larger memories and screens. Table 2.5 lists the power consumers of a mobile phone when it is in full power-on processing mode [157]. Using Wi-Fi for mobile Internet browsing can drain the battery within hours. If the Wi-Fi is not used, power is demanded by CPU and memory modules. In the Table, "other" accounts for subsystems like screen and keypad background lights. In order to conserve battery, mobile phones use different optimizations like turning off the CPU and screen when their usage is not required. The approach may affect the software development for mobile devices (see Section 2.2.3) but is necessary for power savings in more ever demanding mobile phone hardware.

ARM (Advanced Reduced instruction set computer Machine) processors used by Nokia devices running Symbian mobile operating system work at a little over 100MHz from the first Symbian device (Nokia 7650) to more than 600 MHz for the latest devices (Nokia 6710, Nokia 6720) running the latest Symbian OS [72, 14]. Other smartphones are running in a similar range of 600+MHz (for example Apple iPhone [11] runs with 624MHz). There is even an attempt having 1GHz CPU speed on the mobile device - Toshiba TG01 [115] running Microsoft Windows Mobile operating system.

The expected minimum memory found on mobile devices can be concluded from specifications like CLDC for JME (for more see Section 2.2.2). JME configurations specify requirement for 160-512kb of total memory including minimum of 160kb of ROM and 32kb of RAM. In reality the mobile industry fragmentation applies and available memory can in some cases reach to tens of megabytes [49, 155].

Overcoming the mobile hardware constraints

With the recent rise in the mobile Internet usage (see Section 2.1.3) it is obvious that hardware constraints are an issue. Mobile users are expecting their Internet browsing mobile devices to be easy-typing and with big colorful high-resolution screens. The mobile phone will hardly be as capable as current PC platforms. Some innovative technologies can be promising. Though, Apple iPhone introduced the first successful touchscreen[6] device and have become a relatively popular Internet browsing mobile device. Touchscreen technology itself tries to eliminate both of the input and output problems of the hardware constraints. As the screen can be used as an input interface the area won from getting rid of the keypad can be used for a bigger screen itself. Apple iPhone uses a 320 x 480 pixels 3.5 inches touchscreen and has no separate keypad. Many other mobile device vendors are shipping devices alike. Still, the generic approach is to fit both a good input and output device onto the same mobile phone as touchscreens may be uncomfortable in some cases (for example, in winter with gloves or when user interface requires accurate pointing). As for the mobile software developers there seems to be no leverage on a standardization of the input and output hardware in the mobile industry.

2.2.2 Software constraints

Mobile software constraints are mainly due to the software running inside the mobile device. Related either to the operating system (OS) or some other 3rd party software. Unlike the PC world, no single operating system dominates the mobile device industry. As the mobile OS is directly related to the hardware manufacturing process, each mobile device manufacturer has its own flavor of software running on the mobile device model. Some common sets of features are being formed on the more advanced mobile operating systems like Symbian, Apple OS and Android, although, the latter two are immature operating systems and still evolving (in Section 2.2.2 see the Apple and Symbian OS update differences since the release). Defining something concrete about their stability is currently too early.

There are hundreds of vendors that offer mobile embedded software. It is not easy to get the software onto the mobile device in pre-release i.e. before the mobile device have been released. VisionMobile [118], a market analysis and consulting company, lists a so called "100 million club": a watchlist of software companies whose products have been embedded on more than 100 million cellular handsets before they are released [164]. Very few software companies have managed to overcome the commercial and technical challenges inherent to the mobile industry. Based on the research by VisionMobile, only 25 products from 23 companies have shipped on more than 100 million handsets as of June 2008. These figures emphasize the complexity of the mobile market at a time when over 6 billion handsets have shipped as of the first half of the year 2008.

[6] a display which can detect the presence and location of a touch within the display area.

The "100 million club" confirms our statistics in Section 2.1.3 of Nokia being one of the most popular devices as it ships with Symbian, the S60 operating systems and the S40 application framework. However, they are still no match to the amount of real-time operating systems found on mobile phones.

A software constraint in mobile industry is defined by two main factors:

1. The diversity of the software that runs the mobile devices, and
2. specific software bugs within these software.

The software that runs on a mobile device is categorized into two main groups:

1. Mobile real-time operating systems.
2. Mobile operating systems.

Both of them may support development of a 3rd party software on top of it. It is done either directly on top of the operating systems or using an application environment.

Mobile real-time operating systems

A Real-Time Operating System (RTOS) is a multitasking operating system intended for real-time applications. Such applications include embedded systems (programmable thermostats, household appliance controllers), industrial robots, spacecraft, industrial control, and scientific research equipment [95]. In mobile industry these systems are used on mobile devices and are generally closed, proprietary and customized for a specific mobile model.

Two main mobile RTOS's are OSE (Operating System Embedded) [81] and Nucleus OS [75]. VisionMobile estimates over half of the mobile devices shipped with RTOS's by June 2008 [164]. As being closed and very customized, the software development on these devices cannot be made on top of the operating system itself. Generally, application environments shipped with mobile RTOS's are used for 3rd party software development and the software constraints are inherited from that layer of the software (application environments will be discussed later in this Section).

Mobile operating systems

A mobile operating system, also known as, a *mobile OS*, a *mobile platform*, or a *handheld operating system*, is the operating system that controls a mobile device - similar in principle to an operating system such as Linux or Microsoft Windows that controls a desktop computer. However, they are currently somewhat simpler, and deal more with the wireless versions of broadband and local connectivity, mobile multimedia formats, and different input methods [68]. Generally, they support 3rd party software development and have a bigger size footprint than mobile real-time operating systems. This also is a reason for the mobile OS devices offering more advanced capabilities and generally called smartphones for their often PC-like functionality.

Based on the current status of the market (see Section 2.1.3) the most common mobile operating systems have the following attributes:

- Symbian OS
  - current market share within smartphones is around 43%

  - proprietary software, owned by Nokia through Symbian Foundation [109].
  - licensed under Eclipse Public License (EPL) [28] (free software, non-GPL, copyleft, since June, 2008).
  - runs officially on ARM (Advanced Reduced instruction set computer Machine) processors.
  - first / latest stable release: 2001 / 2007
  - releases so far: 9
  - number of supported models: approximately 250 different phone models since the formation of Symbian [109].
  - constraints to 3rd party applications development:
    * limited distribution from application signing requirements (see Section 2.2.3).
    * software development kit (SDK) requires Microsoft Windows PC operating system.
    * programming language: a constrained non-ANSI C++

- Apple iPhone OS (also known as OS X iPhone)
  - current market share within smartphones is around 33%
  - proprietary software, owned by Apple Inc. [13], closed source.
  - runs only on ARM version 6 family-based Apple iPhone mobile devices.
  - first / latest stable release: June, 2007 / January, 2009
  - releases so far: 15
  - number of supported models: 2 main models
  - constraints to 3rd party applications development:
    * one-time fee to enter iPhone Developer Program for acquiring development certificates.
    * software development kit (SDK) is dependent on Apple PC operating system Mac OS-X [12].
    * dependency on Mac OS X introduces dependency from Apple Mac PC .
    * application distribution is controlled and limited by Apple's Application Store where restrictions may apply due to the content of the application.
    * programming language: Objective-C

- BlackBerry (RIM) OS
  - current market share within smartphones is around 10%
  - proprietary software, owned by Research In Motion (RIM) [96], closed source.
  - runs only on RIM BlackBerry mobile devices.
  - number of supported models: approximately 60 different models excluding the early pager-only devices.
  - constraints to 3rd party applications development:
    * development is done using either JME application environment, and/or

    - * RIM's proprietary Application Programming Interface (API) where some of the API requires certification of the application by RIM.
    - * Programming language: Java

- Microsoft Windows Mobile
    - current market share within smartphones is around 7%
    - proprietary software, owned by Microsoft [64], closed source.
    - first / latest stable release: June, 2003 / May, 2009
    - releases so far: 7
    - number of supported models: approximately 300 devices from which many are the same operator branded models.
    - constraints to 3rd party applications development:
        - * SDK is based on Microsoft Win32 API - dependency from Microsoft Windows PC operating system.
        - * application distribution is not directly limited.
        - * programming language: C++ (based on Microsoft Win32 API)

There are also other mobile operating systems missing from the list like Palm, Google Android OS and Linux-based mobile operating systems. The scope of this thesis does not allow to fully compare and describe all of them. Symbian OS will be described more specifically as a potential mobile operating system platform for the F2F Computing mobile framework (see Section 2.2.3).

The list above emphasizes the generic problem of the mobile OS fragmentation found in the mobile industry. All listed operating systems do support software development for 3rd parties. However, there is no unity between the different OSes. Difference is found in almost every aspect of the operating systems from the legal issues, distribution to the details of the coding where the different programming languages and dependencies are applied. Additionally, there are devices still being sold and used with the previous versions of the operating systems introducing multiple invariants within this fragmentation. The only unity to be found is that they all end up running inside a mobile phone. What happens in between is a very colorful but diffuse world. To overcome this constraint a 3rd party mobile software is usually targeted to a specific mobile operating system (like Symbian) or in some cases application environments are used.

Application environments

Application environment is a set of APIs and a programming model to be followed in order to create software for a mobile device that is running the application environment. The advantage of an application environment is hiding the specifics of the platform it is running on. Some variance expressed through the software bugs does exist. In general the software developer should not be concerned about the concrete mobile operating system or the real-time operating system it is running on. An application environment API should hide this diversity introducing a unified set of APIs.

Application environments do not leverage the constraints found in hardware (see Section 2.2.1) and operator domain (see Section 2.2.4). These constraints need to be

taken into account when using application environments. Additionally, application environments tend to introduce their own set of constraints to do with limited access to the underlying operating system and are intended to be generic within a fragmented industry. VisionMobile [164] lists two of the most popular application environments available for the mobile devices - Adobe Flash Lite [2] and Java ME [53].

Adobe Flash Lite Adobe Flash Lite is a software for viewing animations (with vector based graphics support), movies and other interactive audio-video components for the mobile device users. It is a lightweight version of the The Adobe Flash Player [4] that is generally used inside the PC-based web browsers. For mobile it can be used as either a separate application environment or though a mobile web browser[7]. The applications are developed using a proprietary language called ActionScript and other Adobe's Flash tools (for more, please refer to the Adobe Flash Player official website). Flash Lite is very popular in Asia, getting its ground in Europe and not very spread in the United States.

Multimedia content is very demanding regarding hardware requirements. When Flash Lite applications are rendered, poor graphics performance and sound handling can be an issue due to the low screen resolutions, processing power and memory limitations (see Section 2.2.1) . If more capable devices appear the power issue could be reduced.

Java Micro Edition Java Micro Edition (JME), sometimes also referred to as Java ME or by its previous name J2ME (Java 2 Micro Edition) is a specification of a subset of the Java [55] platform aimed at providing a certified collection of Java APIs for the development of software for tiny, small and resource-constrained devices. Mobile phones fall into the category of resource-constrained devices and JME is present on millions of mobile devices and still shipping as the platform is well positioned within the mobile industry [164].

Subset of the Java API means not only limitations to the API but also limitations to the power of the language. The current version of the JME is based on the Java version 1.3. This means that most of the new features of the latest Java (like generics and simpler syntax based on them) are not accessible to the JME programming language. The programming is basically still done like it was in year 2000. This problem is trivial if the code needs to run in an optimized mode within the constraints of hardware and limitations applied by the JME application environment.

The JME is divided into *configurations*, *profiles* and *optional APIs* [147]. Configurations specify a Java Virtual Machine and some set of core APIs for a specific family of devices. Configurations depend on the memory and processor constraints of the device defining the limits of the hardware. Two configurations exist:

- Connected Device Configuration (CDC) - a device with:
  - minimum of 32-bit CPU,
  - minimum of 512kb of read-only memory (ROM),
  - minimum of 256kb of random access memory (RAM) (for the Java platform), and

[7] this may still differ based on a model and in some cases only one of the modes is supported

  - a network connection[8]

  CDC is mostly applied to limited devices like television set-top boxes, car navigation systems and high-end Personal Digital Assistants (PDAs).

- Connected, Limited Device Configuration (CLDC) - a device with:

  - minimum of 16-bit CPU,
  - 160-512kb of total memory including:
    * minimum of 160kb of ROM, and
    * minimum of 32kb of RAM (for the Java platform), and
  - a network connection.

  CLDC is mostly applied to limited devices like mobile phones, pagers, PDAs. Two versions of the CLDCs are applied to mobile devices:

  - CLDC 1.0, with limitations [126]:
    * floating point math and data structures are not supported.
    * weak references are not supported.
    * no native methods.
    * no custom classloaders.
    * includes limitations listed in CLDC 1.1.
  - CLDC 1.1, with limitations [129]:
    * serializable interface is not supported.
    * parts of the reflection capabilities of the Java standard edition are not supported.

Profiles define a set of APIs and specifications necessary to develop applications for a specific family of devices. A profile defines the set of APIs for a configuration. The most common profile in the mobile phone industry is the Mobile Information Device Profile (MIDP). Two versions of MIDP exist:

- MIDP 1.0 [127]. Additional to the limited set of Java SE, a device specific API is added for the:

  - user interface (UI) elements for user interaction tasks.
  - persistent storage (Record Management System) management.
  - lifecycle management of the JME program (initializing, pausing, resuming, exiting).
  - connection API (requires HTTP only).

  The first launch of the JME-based MIDP 1.0 phone was in April, 2001.

- MIDP 2.0 [128]. Extends MIDP 1.0 with adding:

[8] the device needs to be "connected" as defined by the configuration.

- multimedia (sound) playback.
- 2D graphics enhancements.
- authenticate API to support Public Key Infrastructure features like:
  - secure HTTP support, and
  - socket connection.

Modern mobile devices have much more features than the "outdated" MIDP profiles define. MIDP specifications themselves define many requirements as *SHOULD*. MIDP 1.0 defines *SHOULD* as:

> "Indicates a recommended practice. There may exist valid reasons in particular circumstances to ignore this recommendation, but the full implications must be understood and carefully weighed before choosing a different course."

Additionally the term *MAY* is used and defined:

> "Indicates that an item is truly optional."

Specification implementations using "should" and "may" within millions of mobile devices created a fragmentation between the implementations. Additionally, some mobile manufacturers implemented more than required by the MIDP. For additional features, not listed in the "outdated" MIDP, many optional APIs were defined, developed and added to the mobile devices. These, for example, add support for SMS messaging, access to the address book of the mobile phone, more capable file access to overcome limitations of the standard Record Management System, Bluetooth communication support, more advanced audio API and even a 3D graphics API. JME fragmentation was created.

A newer approach is developed to leverage the fragmentation created by MIDP 1.0/2.0. The goal is to gather all the existing specifications under one set of APIs provided for the mobile devices. Mobile Service Architecture (MSA) [52] defines a backward compatible subsets of APIs that a phone needs to implement. Already three subsets are defined - MSA 1.0, 1.1 and 2.0. MSA is already being considered and applied by some of the mobile manufacturers like Nokia and Sony-Ericsson. Even if the MSA is being implemented on the device it may define additional APIs outside of the MSA. The fragmentation is retained. Additionally MSAs with APIs defining requirements using "should" and "may" definitively will not eliminate the mobile industry software level fragmentation issues. Nevertheless, targeting application environments for mobile software application may give a wider audience to the software. JME is still a very attractive platform to reach millions of mobile devices and users. All this comes with a price of limitations within the platform and fragmentation of the features.

Overcoming the mobile software constraints

Fragmentation of the mobile software needs to be accepted by the mobile software developers. It is clear, that not all devices can be reached with the same source of the software being developed. Depending on the mobile application features a concrete audience needs to be targeted. The widest supported platform that meets these customers is to be picked. Smartphones, running mobile operating systems, are generally

used by people who are more business-wise or like to do a lot of data-based communication and get more PC-like features from their mobile devices. Still, newer devices from voicecall-only to more multimedia-savvy are able to confront the smartphone class. If a "down-to-iron" approach is not required, then more generic application environments can be targeted. Fragmentation of the APIs can be leveraged with managing databases that describe the different APIs found on a mobile device. Good examples are WURFL [155] and J2ME Polish [49]. They definitely are not the solution for the whole mobile industry fragmentation issue but can be a solution for the developer. In general, the fragmentation is responded with fragmentation within the mobile application development itself, where for an application many separate compilations of the same program are created. Managing these variants is the complexity found in the mobile software development.

2.2.3 Symbian mobile operating system constraints

Symbian mobile operating system (OS) is one of the most popular operating systems found within smartphones. Its current share is accounted for more than 40% of the smartphones found at the time of writing this thesis (see Section 2.1.3). Symbian is installed on approximately 250 different phone models since the first release of the Symbian operated mobile device in 2001. The software is proprietary and owned by Nokia through Symbian Limited. Some of the software parts are licensed under Eclipse Public License (EPL). Symbian officially runs on ARM (Advanced Reduced instruction set computer Machine) processors. The OS is used by many mobile manufacturers like Nokia, LG [59], Motorola [70], Samsung [98], Sony Ericsson and NTT DoCoMo [74] devices in Japan.

Operating system

The Symbian OS was designed for use in small battery-powered devices with extensive communications capabilities. Its key design features include [142]:

- Performance - designed to maximize the battery life through careful device-specific power management.
- Multitasking - telephony, messaging and communications are fundamental components designed to work in parallel.
- Object-oriented software and highly modular architecture.
- Memory management optimized for embedded software environment - very small executable sizes and ROM-based code that executes in place.
- Runtime memory requirements are minimized.
- Security mechanisms for enabling secure communications and safe data storage.
- Application support for an international environment, with built-in Unicode character sets and ease of localization.

Symbian OS has a microkernel architecture. It contains a scheduler, memory management, and device drivers. Other services like networking, telephony, or filesystem support are placed on top of the microkernel.

Symbian OS applications are event-driven rather than multithreaded. Multithreading is possible and is used with the Operating System, but it is generally avoided in applications, because it potentially creates several kilobytes of overhead per thread: avoiding threads avoids need for any context switching; avoiding context switch requirement introduces low overhead on the overall application within the constrained hardware. Event-driven applications include pre-emptive programming style and needs to be followed when programs are written for the Symbian OS [142].

User interface modules are written on top of the Symbian OS and are provided for separate platforms like NTT DoCoMo's MOAP(S), Nokia S60, Nokia S80 and Nokia S90. The user interface modules use the Symbian OS microkernel to access data management, communications, graphics, multimedia, security, personal information management (address book, phone numbers), SMS messaging, data synchronization and internationalization libraries. As the NTT DoCoMo's MOAP(S) is unreachable in the context of this thesis, and Nokia S80 and almost non-existent Nokia S90 are targeting high-end communication devices, the interest of thesis balances around Nokia S60 platform.

Nokia S60 platform

Series 60 (S60) Platform builds on top of the Symbian OS, complementing it with a configurable graphical user interface (UI) library, with a suite of applications and other general-purpose libraries for the software developers. The suite of application includes for example JME and Flash Lite application environments (see Section 2.2.2). Additional application environments can be installed to the device like Python for Series S60 (PyS60) [91]. PyS60 allows writing mobile applications using the Python programming language and enables to port programs written in Python onto the Symbian S60 platform.

Since the launch of the S60 device there have been four major releases [130][142]:

- Series 60 (ver. 0.9 and 1.0)
  - ver. 0.9 released in 2001
  - underlying Symbian OS ver. 6.1
- Series 60 2nd Edition (ver. 2.0, 2.1, 2.6, 2.8)
  - ver. 2.0 released in 2003
  - underlying Symbian OS versions:
    - 7.0s for S60 2.0, 2.1
    - 8.0a for S60 2.6
    - 8.1a for S60 2.8
- Series 60 3rd Edition (ver. 3.0, 3.1, 3.2)
  - ver. 3.0 released in 2005
  - underlying Symbian OS versions:
    - 9.1 for S60 3.0

    * 9.2. for S60 3.1
    * 9.3 for S60 3.2

- S60 5th Edition
    - ver 5.0 released in 2008 with S60 version 5.0.

The new releases followed the new features like cameras and Global Positioning System (GPS) appearing on the devices. With new features new UI experience and software was expected from the S60 platform. S60 depending on the underlying Symbian OS required new features from it and version updates were applied.

Generally, migrating a Symbian program to a newer Symbian releases is not an issue. Programs compiled for the older platforms worked in the newer ones or a recompile was the fix. A major exception came with the S60 3rd Edition. It introduced a binary break for the older S60 programs. Older applications need to be updated, and in some cases modified and recompiled using the new tools provided in the S60 3rd Edition Software Development Kit (SDK), in order to run on S60 devices 3rd Edition devices. The reason for this was the reintroduced security model and mandatory code signing process for the S60 3rd Edition and upwards platforms [130]. Nevertheless, S60 3rd Edition is currently the *de facto* S60 platform. It will evolve as the only UI platform for Symbian (the process of open sourcing the Symbian platform is discussed later).

Nokia S60 3rd Edition development

As already mentioned, S60 supports application environments like JME, Flash Lite and Python. Its core, still, is to support 3rd party software development on top of Symbian OS. Symbian OS is written largely in C++. Each version of the S60 platform provides its own specific C++ Software Development Kit (SDK). The S60 SDK provides documentation, tools, and sample code to assist developers, along with a Microsoft Windows-hosted emulator. The SDK is essential for developing, testing, and debugging Symbian C++ applications [142]. The official dependency of the SDK on the Microsoft Windows operating system is a constraint for mobile developers used to other operating systems. Another limitation is that the SDK emulator only emulates but not replicates the mobile device. In some cases it even represents fewer constraints (memory, access rights) compared with the real mobile device. Developing, testing and debugging with the SDK means that the application needs to be re-tested on the device itself in the real environment to make sure it is behaving as expected. In the fragmented mobile market, it is a separate complex task with many devices and operators being present. Nevertheless, existing tools with SDK help a lot in the Symbian OS application development process.

Generally, the learning curve for the Symbian C++ programming language is steep. Symbian OS contains hundreds of classes and thousands of member functions [130]. Another set is applied by the S60 framework itself. Programs written for the Symbian operating systems expect the code to be as optimized as possible as it will be run inside a constrained environment. The requirement applies limitations to the programming language itself. The API is very declarative as there is a strong emphasis on conserving resources. Even at the primitive type level. For example, to define an *int* primitive a Symbian special type has to be used - like *TInt* for signed integer type of the natural machine word length that in all implementations is guaranteed to be at least 32 bits.

These types are then mapped to correct primitives based on the targeted platform in compile time. This is only a fraction of learning curve required if overlooking the Symbian-specific programming idioms such as descriptors and a cleanup stack (created objects need to be manually removed for memory leaks), event-based programming with active objects, missing of the non-standard throw-catch exception mechanism and specific naming convention where the name defines the object's memory management. All this in addition to the mobile specific features of off switching CPU when applications is not active or even putting on hold or shutting down the application when mobile device requires more resources for more essential tasks any time (like incoming call). This can make even relatively simple programs hard to implement. An ideological question arises: are the techniques of software development designed for the devices from the last century are still applicable and maybe cause unnecessary complexity in the source code for today's more powerful devices? The technique requires for the programmers to follow error-prone routines like cleanup stack instead of implementing application specific features. Fortunately, tools like S60 SDK and supplementary Integrated Development Environment tools like Carbide.c++ (an Eclipse variant for Symbian OS development) [33] provide some help in the Symbian mobile software development.

Porting existing C/C++ program onto Symbian requires a major rewrite of the application (refer the *int* example above). Fortunately, Symbian is trying to boost software development for Symbian OS and tools are developed for porting existing C/C++ code onto Symbian OS. Open C/C++ for the S60 platform [34] enables porting of an existing C/C++ programs to Symbian mobile platform wrapping the complexity of rewriting the source. The APIs helps developers who lack knowledge of the Symbian OS and S60 APIs to contribute applications to Symbian OS.

Once the application is ready for deployment, another issue is faced - the revised security model and mandatory signing of the applications created for S60 3rd Edition and upwards. This update introduced a concept of *capabilities*, entities of protection. A functionality (i.e. an API) within Symbian OS has a capability associated with it if it needs to be protected. Code requiring to use that functionality must be guaranteed to use the API in a safe way. Code that needs to access capability-protected functionality must go through the process of *authorization*, in order to gain the privilege of using the capability. Authorizing a capability effectively confirms that the code is trustworthy enough to use the functionality protected by that capability [160].

A phone manufacturer may choose to allow the user to authorize selected capabilities. These capabilities are called the "user-grantable" set. If a capability has been included in the "user-grantable" set, it is possible for the mobile device to prompt the user to request that an application be granted to use capabilities requested by the installed application. User-grantable capabilities are:

1. Capability for sending or receiving information through USB, IR, and point-to-point Bluetooth profiles.

2. Capability to access location information of the phone (GPS).

3. Capability to dial a phone number, sending SMS message or connect to network.

4. Capability to read and write confidential data, such as user contact information.

5. Capability to access live information like recording audio or using camera.

However, it should be noted the set of "user-grantable" capabilities may vary according to manufacturer and/or device. Requiring more capabilities will require signing the application through Symbian Signed [110] process. Other capabilities include:

1. Capability to kill any process in the system or switching the state of mobile (turn the phone off, move to offline mode).
2. Capability to read and/or write confidential network and phone data like:
   (a) read information about the current network, the identification information from the current mobile base station (Cell ID), or
   (b) reading the International Mobile Equipment Identity (IMEI) code that is unique identifier of the mobile device, or
   (c) manage device settings like reading information about the installed applications.
3. Capability to access logical device drivers (software layer of the drivers).
4. Capability to emulate user actions (like key presses) and send them to a application.
5. Capability to create a trusted user interaction (UI) session, and, therefore, to display dialogs in a secure UI environment.
6. Capability to access critical multimedia functions such as direct access to associated device drivers and priority access to multimedia (sound, camera, video, etc) APIs.
7. Capability to manipulate Digital Rights Management (DRM) [26] protected content.
8. Capability to modify or access network protocol controls (for example, forcing all existing connections on a specific protocol to be dropped, changing the priority of a call or changing the type of the connection from Wi-Fi to mobile broadband).
9. Capability to directly access all communications equipment device drivers (Wi-Fi, USB and serial device drivers).
10. Capability to access the file system administration operations that affect more than one file or directory - managing or overall file-system integrity and behaviour like mounting and unmounting a drive partition.
11. Capability to grant access to system specific directories of the device.
12. Capability to grant visibility to all files in the system and extra write access to files that are owned by the installed applications only (so called private files).

If the application is intending to require any of these capabilities a Symbian Signed [110] signing process has to be applied to the application. Signing is the process of encoding a tamper-proof digital certificate into an application's installation file. The certificate identifies the application's origin, and grants access to those capability-protected APIs that are requested by the installable application. To get the application signed, it

has to be verified by an accredited testing house [111]. Every change to an application needs to be re-verified. The signing process includes yearly and per-signing costs for the developer of the application. The process of Symbian signed being costly has created some disturbance around the Symbian OS open source initiative.

Symbian S60 as an open-source platform

The Symbian OS is not fully open source yet. The Symbian Foundation, a non-profit organization was formed when Symbian Ltd. was acquired by Nokia in June, 2008 [109]. Its main aim is to unify Symbian, S60 and MOAP(S) and open-source them in the next two years after it has expected to become fully functional in first half of 2009. To become a member of the foundation, an annual fee (that is a non-symbolic 1500$) is expected. Currently, only legal bodies (a non-individual) can request the membership. Contributions to and licensing the Symbian platform will be accepted from the foundation members only. The model of the Symbian Foundation and existing signing constraints of the Symbian S60 3rd Edition are questioning the "open source"-ness of the Symbian OS platform now and in the future. Limitations already exist in process of copying and public improvement of the existing program through the Symbian Signed required signing process of the applications for the S60 3rd Edition programs.

2.2.4 Constraints applied by operators

A mobile network operator, also known as *mobile phone operator* (or simply *mobile operator* or *mobo*), *carrier service provider*, *wireless service provider*, *wireless carrier* or *cellular company*, is a telephone company that provides services for mobile phone subscribers [67]. Historically they are enterprises who grew out from companies building up the mobile networks. Nowadays they have evolved into industrial giants playing major roles in the mobile industry.

The operator's main interest historically is to keep subscribers using their core product - the voice-based services. The ongoing shift away from voice-intensive cellular technology to data-intensive mobile broadband is a significant change within this model (see Section 2.1.3). Constraints are applied to make sure that clients are loyal to the operator. These constraints are sometimes referred as *walled gardens* and generally fall into two categories:

- customizing a mobile device to solely functioning only under the operator, and/or
- constraining the network's features to ban competitive services.

Customizing mobile devices

Customizing the mobile device generally includes some means of making sure that the mobile device cannot be used with other operators restricting it to work under a certain network operator only. SIM-locked [100] devices can be unlocked. This still may not remove the possible software level constraints set by the operator and is usually in breach of the agreement with the mobile device provider. Constraining software is the second way that operators are using to make sure that only their services are being used. In the context of software development, changes made to the software within the mobile device is a problematic issue. The practice increases the software fragmentation

within the mobile industry (see Section 2.2.2) and in some cases it can be very complex to detect because the restrictions are not widely published.

Since the introduction of the Voice over Internet Protocol (VoIP) applications that use data to carry speech over Internet-enabled networks are being seen as a danger to a mobile operator's core business. Mobile devices may be customized by the operators in order to avoid these kind of applications. For example, mobile operators Orange [78] and Vodafone [119] have been publicly announcing to disable VoIP in Nokia N95 devices and that the move is not a deliberate attempt to defend voice revenues [79]. Another way of constraining the mobile application is to customize either mobile device operating system or its application environments. This approach is used when a generic feature is expected to be limited. Data services over mobile networks is a generic feature that falls into this category.

JME application environment applications belong to a protection domains when running in the mobile phone. The domains determine which permissions are granted, which are denied, and which ones must be deferred to the user's judgment. This model is intended to control applications downloaded from the "insecure" Internet and is called mobile application certification. JME certification, similar to the model used in Symbian operating system (see Section 2.2.3), is based on the the Public Key Infrastructure where some third party trusted entity certifies the application for the mobile devices to trust. Every certification requires passing a set of tests done by testing houses and is a payed service for every application and its update making it rather costly process [54]. The concept of protection domains is deliberately vague, leaving MIDP vendors (the operators and mobile manufacturers) with considerable latitude in their implementation [147]. Some mobile operators, like AT&T [15], provider of both local and long distance telephone services in the United States, is known to customize the JME application environment exactly in that area. AT&T requires the JME applications to be certified in order for them to access some of the JME APIs. These APIs include Bluetooth and socket-based network connections, file and personal address book access, SMS sending features, location (GPS) and audio recording features [15]. Restricting socket connection may paralyze the workings of many data-heavy applications like VoIP or any other multimedia-related application. Without the modification in the JME's application environment, accessing these APIs would ask the user permissions with a dialog where user's judgment is being consider and the application will have access to the APIs. AT&T's signing policy has taken this decision away from the mobile device users and made it its own. With this move, it has made decision to close this market to many freeware JME applications that rely on the restricted APIs.

Constraining the networks

Operators have historically constrained their networks. First Wireless Application Protocol (WAP) gateways [120] where walled gardens - very closely controlled mobile Internet portals that the wireless mobile devices accessed every time when they opened the mobile Internet. The real constraint applied when the device was additionally software constrained and the device browser's home address could not be accessed and/or changed. As the mobile Internet usage is in the rise (see Section 2.1.3) and mobile users can access virtually whatever web page they're interested in, then this approach is abandoned and in some cases replaced with more generic one by limiting the maximum download size per request.

The Netsize Guide 2009 [137] lists the limits for maximum WAP gateway download

size used by operators in Poland (100kb), Ireland (300kb) and in the homeland of one of the biggest mobile device manufacturer Finland (10-15MB). Tens of megabytes of web content per request may be enough, but only hundreds of kilobytes can be an issue. 100kb limit on download may limit the installation of an application larger than the 100kb. Mobile application with animations, sounds and other multimedia-heavy content can simply overcome the 100kb size-limit.

Another way to bind the users to the operator's core business is to make sure that applications that compete with operator's main business model are constrained. An example in the previous section to limit a certain popular mobile device by an operator can become a generic approach for others - limit the whole VoIP service within the network [42]. The future of the VoIP popularity and worldwide allowance in the mobile industry is still to be seen as it is with opening the walled gardens of operator networks.

Overcoming the operator constraints

Operator constraints is the most complex and closed problem domain in the mobile industry. Generally they are not very "loudly" discussed because of their limitative nature for the industry that should be booming of innovation and new ideas. The information is circling in the mobile community either through independent developers or forum sites. Some operators, like AT&T, are willing to publicly define the limitations. As with forum sites, the validity of the source can be questioned but in some cases can be the only hint to the existing walled gardens. Fortunately the operators are showing some signs of becoming more open due to the social requirements push [137]. In current state, the nature of this thesis is to accept the operator constraints "as is".

2.3 Grid computing in the mobile context

Grid computing means that several computers are used as one. It networks single computers, servers, supercomputers, clusters and special devices into a global resource [140]. Usually, the Grid is used to solve scientific or technical problems. Typically they require heavy computational power. That is achieved by distributing the task to many computational peers and solving the calculation problem generally in parallel. Within the context of Mobile Computing [158] the possible *Mobile distributed computing*, or *Mobile Grid* is becoming an interesting mobile distributed computing platform. Mobile Grid means that movable wireless devices are integrated into traditional wired Grid through a wireless channel to share Grid resources (CPU power, storage capacity, instrument, devices, data, software, etc), meanwhile, mobile devices can provide service or resource to Grid users, such as PDAs, cellular phones, handsets or wearable computers, laptops with GPS service, mobile service, etc. [139].

Billions of mobile phones exist and are connected to the network though billions of subscriptions (see Section 2.1.3). Although, relatively limited as a single device, the total computational power has potential. To implement mobile distributed computing using middleware for the mobile devices is not something new due to the fragmentation found in the mobile industry. Mobile devices are not shipped with Grid-aware libraries. The middleware integrates the tools of Grid computing – e.g., data communication, directory and service distribution, security, resource discovery, and resource allocation – providing the developer a set of reusable and homogeneous resources [151, 141]. Using the middleware requires a backend providing tasks for distributed computing. F2F

Computing framework is a candidate for one of these backends. The missing part is the F2F Mobile Computing middleware for the mobile devices.

Some Mobile Grid research initiatives exist like K*Grid Mobile Grid [56] and Akogrimo [5], both spawn from the PC-based Grid infrastructures. Questions arise: maybe the approach of looking at the Mobile Grid is too PC-based? Or, maybe the mobile devices should not be computational nodes at all? In fact wireless devices, with currently limited resources, would benefit from the opportunity of using a considerable amount of resources made available by all the other devices connected to the network and maybe some mobile devices could still contribute. In any case, when utilizing networked mobile resources challenges arise due to the constraints of the mobile devices described in Section 2.2. These constraints violate many of the assumptions upon which today's distributed systems are dependent on (for example, fast and stable networking).

Another issue comes into light concerning the usability of the mobile distributed computational program on a device that is very personal to the user (see Section 2.1.1) and not just a brick of computational power. The notion applies to both - how the application should be implemented for the user expecting such a personal approach, and how the mobile device limitations are overcome in general? One way to envision, is that Mobile Grids are expected to be either integrated into a stationary grid infrastructures or rather small *ad hoc* in-place-and-time networked systems useful in mobile field work scenarios which occur in fields like archeology, ecology, or biology. Using mobile devices for communication is becoming more natural to some people [143]. Originating from Japan [138], it is certain that mobile Internet based social networking is in rise all over the world [66]. Netsize predicts the usage of the social networks through mobile devices being the second in the next couple of years passing mobile games already in year 2009 [137]. The trend may hint to use a different approach for Mobile Grid like applications too. It may be, that the successful Mobile Grid applications will have to be more social-like application requesting often user direct interference to the workings of the program and providing personally interested content. Mobile device has become just too intimate for anther application running in the background and talking its piece from already limited resources without user's direct intervention into the whole process.

2.4 Roundup

This chapter was explaining issues that can be found by a mobile software developer when creating applications for mobile devices.

At first, it defined the importance of the mobile device in our society. With billions of active users the mobile phone has become a commodity that everyone owns and carries at all times. The chapter gave a brief history of mobile networks and mobile devices. Mobile operators, the companies that funded and run the mobile networks, still dominate whilst mobile device manufactures also play their own role in the evolution of the mobile industry.

The chapter brought out the latest changes in the mobile industry. The most recent major change is that mobile software development has gone beyond the hands of operators and mobile phone manufacturers and become popular with 3rd party developers. Nowadays, everyone can write an application for a mobile phone. The current market state was presented, describing the most popular mobile manufactures, mobile operating systems and mobile devices that in many cases support 3rd party software development.

Mobile phones come in many forms. They encompass everything from a simple voice-only handset to the ultimate communication tool packed with Internet features. In the mobile industry, the existence of many different mobile phones, a myriad of mobile software and different networks come together to create *mobile fragmentation*. Fragmentation leads to constraints for the mobile software development. This chapter described the constraints found at the mobile hardware, software (mainly operating systems), and operator levels. The description of hardware constraints discussed issues arising from the existence of many physically different devices. The topic of software constraints focused on the fragmentation issues found within the most popular mobile operating systems and application environments. Constraints applied by operators were also described, but being one of the most complex and closed domains, the issues are accepted as is. All the descriptions include examples from the real world, alongside possible solutions and/or workarounds to the fragmentation problems.

A knowledge and understanding of mobile constraints is required to port F2F Computing framework for the mobile phones. Software development constraints found in the Symbian mobile operation system were described in a more detailed manner, because Symbian is the mobile platform selected for the F2F Mobile Computing framework port. The selection of Symbian was led by the fact that Nokia (owner and major user of Symbian) is the largest manufacturer of smartphone devices in the mobile industry.

The second to last section defined Grid computing in the mobile context. This section accepted the previously covered topics, and in that light, tried to envision how a successful Grid computing platform like F2F Mobile Computing should be built and used.

Chapter 3

F2F Mobile Computing framework

This chapter describes the work done while creating the F2F Mobile Computing framework. It will:

- describe the implemented features and problems solved in order to overcome the mobile-related limitations presented in the Chapter 2, and
- describe the implementation specifics of the F2F Mobile Computing framework, its basis, dependencies, porting to the Symbian mobile operating system and tools used in the project.

This chapter will be finished with a summary of the achievements set forth in the beginning of this chapter.

3.1 Modification requirements to the existing F2F

F2F Mobile Computing framework features

The required elements of the F2F Computing mobile framework do not differ so much from the initially introduced elements (see Chapter 1). As the problem to be solved is similar, the existing elements should also be the same. It is quite obvious that the mobile version of the F2F Computing framework will introduce re-work and possible constraints (see Section 2.2) to the existing features. The inherited features are thus indented to be preserved in the mobile version of the F2F framework with the notion of "if possible":

- to enable the distribution of the Tasks for the Peers to compute,
- to be lightweight - easy setup, management, usage and extendability,
- enable to use Instant Messaging (IM) communication tools for social trust-based F2F networking channel,
- use Peer-to-Peer (P2P) techniques fostering fast/direct communication between the Peers after the initial IM setup, and
- be supported on as many mobile devices as possible.

Nevertheless, within the expected mobile hardware (see Section 2.2.1), software (see Section 2.2.2) and operator constraints (see Section 2.2.4), the first point is essential to be supported as it is the main feature of the F2F Computing framework.

The F2F Computing framework was originally written in Java. The sole purpose for this was that Java is multiplatform and easily portable. Additionally, the F2F relies on one of the Java's most powerful features in order to execute Jobs and Tasks within remote Java virtual machines - custom class loading and serialization/deserialization of Java bytecode [53].

Mobile software restrictions in JME (see Section 2.2.2) application environment showed soon, that the missing Java features described above will make the first limitation to the F2F Mobile Computing framework. The support of the main feature, the remote code execution, using Java tools was under the question. As a result, mobile platforms with JME application framework had to be abolished from the potential list of F2F mobile supported platforms. They do not support the required custom class loading, serialization and deserialization of the Java bytecode. The whole refactoring of the core F2F framework became an issue. The F2F core framework had to be refactored into a language that would:

- be supported by a mobile operating system (OS) platform (preferably widely used), and would
- have low learning-curve (for the fast refactoring from the Java-based F2F).

New F2F framework prototyped

At the time of envisioning the F2F Mobile Computing tier in the Q3 2008 Ulrich Norbisrath [154], one of the main authors of the F2F Computing paradigm (see Section 1.1), was bootstraping an idea of lowering the footprint of the current F2F Computing framework to, make it potentially faster and extendable with other languages. As a result, a more clearly feature-differentiated F2F Computing prototype was implemented in C [19] and the Python [92] programming languages.

Figure 3.1 represents the architecture of the F2F Computing framework proposed by Ulrich Norbisrath. It clearly isolates four main layers: *application*, *adapter*, *core* and *communication* layers. Application layer is responsible for the Instant Messaging (IM) Graphical User Interface (GUI) related tasks. In the prototype version this was represented by simple Python-based command-line scripts. The application layer directly uses the IM adapter providing communication services for F2F core and IM GUI for initial communication tasks. After the communication is set up the F2F core is aiming for more faster communication channels like sockets (TCP/IP) or any other communication possibilities found in the Communication layer. The original IM will be an alternative if the others fail or are not available. The prototype version did not implement any of the non-IM communication channels but has defined interfaces for implementing them. Ideally, an Application will be using an F2F Virtual Machine (VM) or Application Programming Interface (API) that will be using the F2F core and its communication layer modules providing F2F Computing capabilities for the Application.

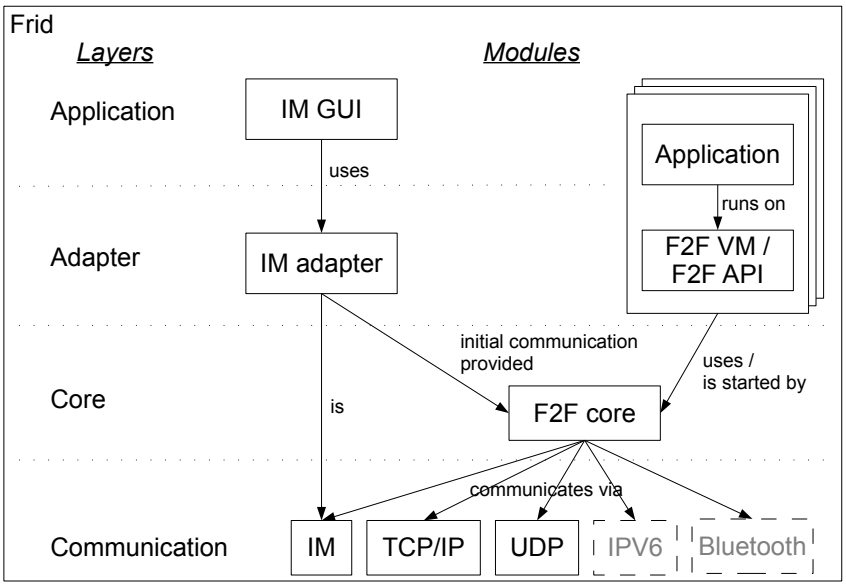

Figure 3.1: New F2F Computing framework prototype

F2F Mobile Computing platform

The prototyped frameworks became a base for the F2F Mobile Computing framework. The C programming language is supported by Symbian Mobile OS (see Section 2.2.3). Using C does not fully solve the problem of remote Task execution - Python will. It supports dynamic execution of the Python code. The interpreter is also ported onto the Symbian with its core feature - the ability to extend the Python modules with the custom 3rd party modules. In our case, this would be the F2F Mobile Computing framework written in C. The remote code execution problem is thereby solved. Also, the obvious choice of the targeted mobile operating system platform for F2F mobile is fixed by this: the Symbian mobile operating system. The minimum requirement for the F2F Computing are satisfied and the technology for the F2F Mobile Computing is fixed:

- The core F2F features will be refactored into Python extended module written in C.
- Python supports remote code execution for the distribution of the Tasks for the Peers to compute.
- Symbian Mobile OS will be the host for the F2F Mobile Computing framework as it has development support for both C and Python.

The porting will be spread across three major reworked layers represented in the Figure 3.2:

1. Client tier (see Section 3.2.4) will be responsible for providing interface(s) for implementing F2F Jobs and enabling access to user interface (UI) components for user interactions.
2. Communication tier (see Section 3.2.3) will be responsible for providing communication features for the F2F Mobile core.

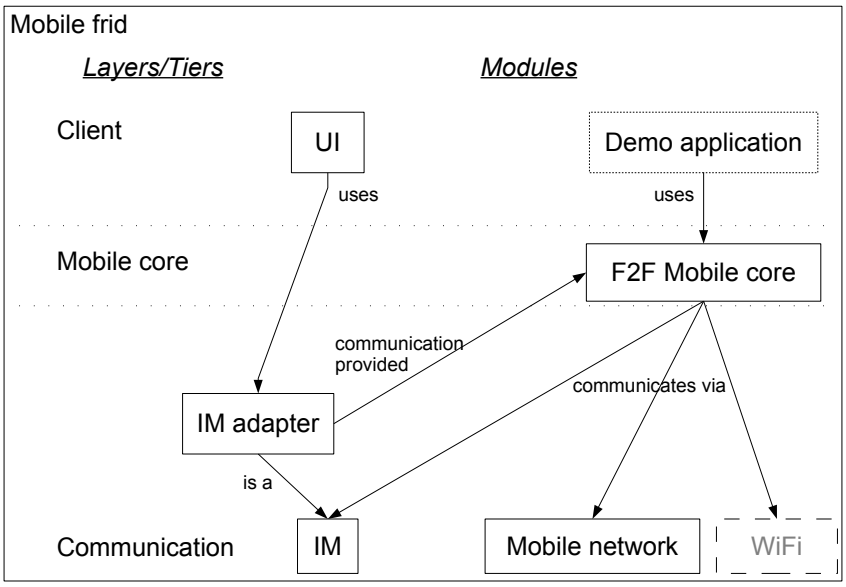

Figure 3.2: F2F Mobile Computing framework

3. F2F Mobile core tier (see Section 3.2.1) will be responsible for the F2F Mobile Computing architecture (see Section 3.1) elements management.

An additional task is creating a demo application running on the top of the F2F Mobile Computing framework (see Section 4).

3.2 Implementation

This section describes in detail the work done implementing the F2F Mobile Computing framework.

3.2.1 F2F Mobile core tier

To set up Frid (see Chapter 1) a certain amount of F2F network management has to be made:

- Peer(s) need to be defined and connected with each other through Group(s);
- Job(s) need to be loaded, parsed and sent out to Peer(s) as Task(s);
- Information of the running Frid needs to be accessible for the Peer(s), Group(s) and Job(s)/Tasks(s) for coordination of the calculation(s), data sharing and Peer(s) management.

A set of standard functions defined in the F2F Mobile core tier (see Figure 3.2) provides the support for the tasks listed above. The communication tier (see Section 3.2.3) will be provided for the F2F Mobile core tier for communication tasks. Client tier (see Section 3.2.4) will use the F2F Mobile core features to compose the Mobile frid and interact with the user. Porting the F2F C code to Symbian mobile operating system introduced limitations described in Section 3.2.5.

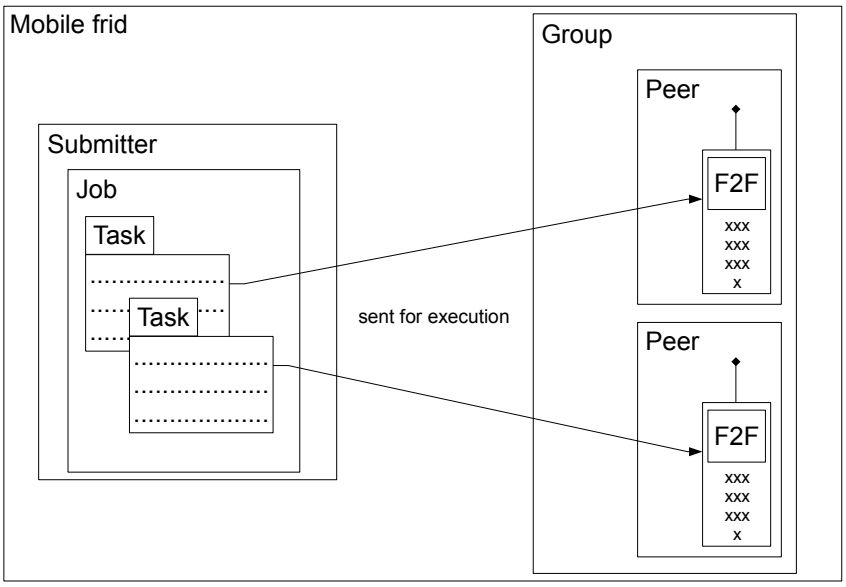

Figure 3.3: F2F Mobile frid

Setting up a Peer

A Peer corresponds to the entity in control of one of the users participating in solving a respective computational Task [145]. A Peer can be a Job parser and submitter, Task solver or both. Every Peer will be identified by a unique identifier (ID) that can be used for direct communication between the Peers. ID clashes are not allowed. Only F2F Peers can communicate to each other and participate in a Mobile frid.

The F2F Mobile core provides a generic function for defining F2F Peers. A unique ID will be generated automatically via a big enough Mersenne Twister random number [149]. This will make sure the absents of the ID clashes. Peer initialization is also provided for the F2F Python tier (see Section 3.2.2).

Creating a Group

Every Peer needs to belong to a Group in which its Job will be computed. Groups are logical collections of Peers sharing some common interest [1]. A Group consists of more than one Peer and a Peer needs to belong to a Group in order to receive Tasks. The F2F Mobile core tier provides functions for:

- creating a named Group (generally done by the Job submitter Peer), and
- adding and removing Peers from a Group;

Peer management in the Group should trigger the request process of adding the user into the F2F Mobile Computing Group. Request process is a process of confirming if the user wants to be part of a (computational) Group. The current F2F Mobile implementation does not have this feature and all the Peers are added to a Group by default (for other future work, please refer to Section 5).

When a Peer belongs to a Group:

[1] For example, IM friends chatting in one chat room form a Group.

- a message can be sent to all the Peers in the Group, and
- a Job will be submitted to the Group and received by all the Peers in this Group.

Group management is also provided for the F2F Python tier (see Section 3.2.2).

3.2.2 Custom Python modules

As described in Section 3.1, the dynamic code execution is required in order to execute the Tasks sent to the Peers. Mobile software restrictions do not allow to have the F2F Mobile core written in the JME application environment (see Section 2.2.2). C was introduced to implement the F2F Mobile core (see Section 3.2.1). Using only C did not solve the ability to send executable code over the wire. This is solved by Python.

Python

Python [92] is a general-purpose, open, community-based, interpreted, high-level (having strong abstraction from the hardware details) language with high-level data types built in (like flexible arrays and dictionaries). A collection of standard modules allows the use of out-of-the-box features like file input/output, system calls and sockets [163]. The set of the default modules alternate depending on the version of Python being used (see Section 3.2.5). Its design philosophy emphasizes code readability. The code syntax and semantics are minimalistic. Nevertheless, Python supports multiple programming paradigms (primarily object-oriented, imperative, and functional) and features a fully dynamic type system and automatic memory management, similar to Perl [84], Ruby [97], Scheme [99] and Tcl [113]. Being a dynamic language, Python is often used as a scripting language [93].

For the F2F Mobile Computing framework, Python was chosen for two major features:

1. dynamic execution of the Python code [163], and
2. extendability with custom modules where new built-in functions or modules can be added to the Python interpreter [162].

Dynamic code execution

When a Job is submitted into the Mobile frid, it will be delivered to the Peers and executed (see Figure 3.3). Depending on the technology used, different approaches can be applied. As described in Section 3.1, the Java-based F2F can use Java's own features of custom class loading (classloader) and serialization in order to achieve remote code execution. The F2F Mobile Computing core is written in C. Introducing Python allows to use Python's dynamic code execution for the similar results [163]:

1. The F2F Job is written as a normal executable Python code.
2. The code is sent unmodified to the Peer - the client. The same source code has to reach the client despite the applied possible packing or communication specific transformations to the source.

3. The received code can now be executed at runtime in the Python interpreter of the Peer unless a syntax error occurs noting that step 1. had syntax error in the initial source. It is expected the Job's source to be valid before sending it to Peers.

The executed code will have access to the running Python interpreter. This may introduce security issues for the Peers running unchecked code. Current thesis is not dealing with the security issues of the F2F Mobile Computing framework and this topic should be applied as a future work (see Section 5). The Current implementation trusts the sourcecode of the Job it receives from the submitter, dynamically loads and executes it.

Custom modules

For the F2F Mobile core to utilize Python's dynamic code execution the Python interpreter needs to be accessed from F2F Mobile core, and vice versa. The solution is to implement the F2F Mobile core as a Python extension module.

Extending Python with C/C++ is a process of writing C/C++ code so that it can be accessed from the Python interpreter. Such extension modules can do two things that can't be done directly in Python [162]:

1. they can implement new built-in object types, and

2. they can call C/C++ library functions and system calls.

Alternatively, it is possible to embed Python into a C/C++ application. In the case of F2F Mobile Computing framework, this is not required and applicable as it will be run inside a constraint Symbian mobile operating system (see Section 2.2.3).

To support extensions, the Python API defines a set of functions, macros and variables that provide access to most aspects of the Python run-time system. The API is equally usable from C++, but for brevity it is generally referred to as the Python/C API [162]. Writing an extension module is a relatively well-understood process, where a "cookbook" approach works well. In short - C code, that is intended to be accessible from Python code, needs to be wrapped with special functions using the Python/C API.

With simple "Hello World!"-like applications, the usage of the Python/C API can be manually feasible. With more complex and larger codebase projects writing a Python/C API layer can be an error-prone task. It involves Python-specific requirements like reference counting for automatic out-of-scope reference deallocation and mapping complex C/C++ types into valid Python types. The goal of the F2F Mobile Computing project was to create a proof of concept F2F Mobile Computing framework (see Section 1.2). With this notion in mind it was clear that the Python/C API layer should not be written or managed by hand (at least in the initial version of the framework) and some kind of generator is required.

The Simplified Wrapper and Interface Generator (SWIG) [101]was used to generate the Python/C API layer for the C-based F2F Mobile core. This layer will enable the F2F Mobile core to be written as a Python extension module. Introducing the Python custom modules to Symbian mobile operating system introduced limitations described in Section 3.2.5.

SWIG

To implement Python custom modules, the Python/C API has to be used [162]. The API gives the Python interpreter the access to the custom C/C++ module, and vice versa. The F2F Mobile core, written in C, is rather complex code. The C headers, that have to be visible for the Python interpreter, all alone contain more than 80 elements (methods, functions and type definitions) that need a Python/C API interface. Manually writing and managing such an interface would be a complicated and an error-prone task. I decided, that the interface should be generated from the existing source-code. A couple of tools exist for generating Python/C API interface from the existing source C/C++ source code.

Cython [25]is based on Pyrex [89]. Its pure task is to simplify writing 3rd party extensions for Python. In general, Cython is a Python with C data types. As Python is also valid Cython code, the Cython compiler will convert it into C code which makes the equivalent calls to the Python/C API. Additionally, as parameters and variables can be freely declared to have C data types, code which manipulates Python values and C values can be freely intermixed, with conversions occurring automatically wherever possible. Reference count maintenance and error checking of Python operations is automated. The F2F Mobile core is implemented purely in C. Cython requires mapping of the C functions and types that will be accessed then from Python. Using Cython requires manual handling of this mapper.

The Simplified Wrapper and Interface Generator (SWIG) is an interface compiler that connects programs written in C/C++ with a variety of high-level programming languages including common scripting languages such as Perl, PHP [85], Python, Tcl and Ruby and non-scripting languages such as C# [19], Common Lisp [23], Java, Lua [86], Modula-3 [69] and Octave [76]. It works by taking the declarations found in C/C++ header files and use them to generate the wrapper code the scripting languages need in order to access the underlying C/C++ code. SWIG may be freely used, distributed, and modified for commercial and non-commercial use [101].

The decision about the Python custom module layering tool intended to use was made by the following facts:

- not have legal restrictions (be open-source),
- have a history of development,
- be an alive project with preferably alive community, and
- be easy to set-up and integrate.

Analyzing both, Cython and SWIG, the latter was chosen to generate Python/C API interface directly from the F2F Mobile core C header files. The main reasoning behind the selection was that compared to Cython, SWIG does not require major manual data structure and function mappings against the F2F Mobile core C header files.

The introduced limitations and required tweaks emerged while porting the generated interface to the Symbian mobile operating system (see Section 3.2.5).

F2F Mobile core as a Python module

In the result of the work described above, F2F Mobile core is implemented as a Python extension module and enables:

- the access to the Python dynamic code execution features,
- a simpler communication tier for the F2F Mobile core (see Section 3.2.3), and
- the ability to write F2F Jobs in Python language (see Section 3.2.4).

Porting the custom Python module to Symbian mobile operating system introduced limitations described in Section 3.2.5.

3.2.3 Communication tier

F2F Mobile core is written as a custom Python module (see Section 3.2.2). The feature:

- connects the code written in the Python interpreter and underlying F2F Mobile C code, and
- does not require the underlying F2F Mobile core to implement communication.

Communication can be implemented as an abstraction and then be provided for the F2F Mobile core (see Figure 3.2). This decoupling allows to have independent Instant Messaging (IM) clients being used for the communication layer. Additionally, for "use P2P techniques fostering fast/direct communication between Peers" mentioned in Chapter 1, a more optimized communication implementation could be provided for the F2F Mobile core. The core itself can stay independent of the specific communication implementation.

When the communication requirements were considered for the F2F Mobile Computing framework the constraints of the mobile context (see Section 2.2) were expected to be utmost problematic. In the field of mobile communication all previously listed mobile constraints may apply: hardware for messaging (see Section 2.2.1), software for any of the networking bugs (see Section 2.2.2) and the most inaccessible one - constraints by the operators (see Section 2.2.4). The limitations in mind the connection implementation have to be:

- a port of a general-used IM client,
- supported by the targeted Symbian mobile operating system,
- simple to install, integrate with Python and F2F Mobile core written as Python extension, and
- preferably not dependent on the 3rd party additional modules that could be required by the Symbian mobile operating system.

As the F2F Mobile core was also working in the context of Python, a possible IM port for Python was considered as a good candidate for providing connection to the F2F Mobile core. This would solve most of the requirements listed above.

Jabber.py

Jabber.py [51] is a Python module for the jabber [50] IM protocol. Jabber is an XML [31]-based communication protocol that is written using the open standard Extensible Messaging and Presence Protocol (XMPP) [122] which is still the core protocol of the Jabber Instant Messaging and Presence technology [31]. XMPP makes the Jabber protocol free and open-sourced [50]. Jabber.py is a free port of the Jabber communication protocol written in and for Python. Jabber.py consists of two main source files:

1. *xmlstream.py* - provides functionality for implementing XML-based network protocols using Python's generic XML modules.

2. *jabber.py* - uses xmlstream.py providing functions that implement the Jabber IM functionality.

This promised rather simple installation and setup process of the communication tier for the F2F Mobile core:

- jabber.py would be used to set up the connection as a Jabber IM client, and

- the connection would be provided for F2F Mobile core that would be using it without knowing its origin or specifics.

Jabber.py was chosen as the IM communication layer for the F2F Mobile core. It allows to use Jabber-based IM communication between the F2F Mobile Computing Peers. The alternative and newer XMPPPY [123] was considered as a replacement. It introduced heavier porting issues as the Jabber.py (see Section 3.2.5) and more lightweight Jabber.py was rolled back in. Future work is expected replacing the current communication layer with more generic implementations (see Section 5).

Python wrapping for communication layer

With Jabber.py the communication layer features where introduced into the Python interpreter. This allows to use the Python dynamic code execution feature (see Section 3.1) without direct involvement from the F2F Mobile core. All the communication will be done outside of it. F2F Mobile core would retain its main functionality of Mobile frid management (see Section 3.2.1). As a result, the F2F Mobile core client (see Section 3.2.4) can use the Python interpreter for writing Jobs (see Section 3.2.4) and *pickling/unpickling* for direct Peer-to-Peer communication.

Python "pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," or "flattening", however, to avoid confusion, the terms used in Python are "pickling" and "unpickling" [134]. When one F2F Mobile Peer wants to sent a Python message (an object) to another F2F Mobile Peer:

1. the data will be pickled,

2. processed by the sender's F2F Mobile core in order to be sent to the correct Peer (F2F-specific communication headers appended to the message),

3. sent over-the-wire using Jabber.py communication layer,

4. received by the receiver Peer over Jabber.py communication layer,

5. processed by the receiver's F2F Mobile core in order to check the Mobile frid message's validity (F2F-specific communication message headers are checked and removed), and

6. handed over to the receiving Peer for unpickling to get the Python object.

Introducing the pickling to the Symbian mobile operating system introduced limitations described in Section 3.2.5.

Using Jabber.py and Python for communication tier

Introducing the Jabber.py as a F2F communication tier on Symbian mobile operating system still introduced limitations described in Section 3.2.5. Nevertheless, using Jabber.py the connection support for the F2F Mobile core is provided. The expectations described in the introduction of this section are sufficiently completed - Jabber.py:

- is a general-used IM messaging client,
- is supported by the targeted Symbian mobile operating system through Python interface,
- is relatively simple to install and use from Python layer, but
- is unfortunately dependent on the 3rd party additional modules for using from Symbian operating system (for more detail see Section 3.2.5).

3.2.4 F2F Mobile Computing framework client tier

F2F Mobile Computing framework client tier will be used by the F2F Mobile Computing framework developer for implementing the F2F Mobile frid applications (see Figure 3.2). The tier provides access to the F2F Mobile core functionality. It will be used to implement F2F Jobs and provides mobile specific user interaction (UI) components for the F2F Mobile frid application in order to interact with the user who's mobile device is running the F2F Jobs.

Implementing Jobs

Initially, the F2F Computing framework is written in Java with SIP communicator being the IM client. Using Java to write a F2F Job(s) was an obvious choice. Especially with the required features of the serialization and remote code execution enabled by Java programming language (see Section 3.1) and SIP Communicator being an extendable IM client to support it (see Chapter 1).

F2F Mobile framework is implemented as a Python extension and access to Python interpreter is supported (see Section 3.2.2). This resulted in the ability to write F2F Jobs using the Python programming language. Using different programming language introduced differences how the F2F Jobs can be implemented.

Java-based F2F Computing framework Job writing characteristics:

- Requirement to follow a predefined interface (the type safety): Java interface has to be implemented and followed.
- Job implementation speed: time is spent on the compilation of the Task(s) into Job.
- Syntax error-proneness of the implementation: syntax is checked by the compiler and this will eliminate the possible remote execution syntax issues.
- Logical error-proneness of the implementation: similar to Python-based F2F Computing framework.

Python-based F2F Mobile Computing framework Job writing characteristics:

- Requirement to follow predefined interface (the type safety): no predefined interface is defined, Job is executed as is.
- Job implementation speed: Job is executed by interpreter, can be developed while F2F framework is running and no explicit restart of the interpreter is required.
- Syntax error-proneness of the implementation: syntax is not checked and remote execution can fail.
- Logical error-proneness of the implementation: similar to Java-based F2F Computing framework.

The F2F Job development in Python can be more faster. With on-the-fly execution of the code the problems encountered in syntax error-proneness is overcome with faster write-and-deploy cycles. Python code writing tools (see Section 3.4) can additionally reduce the syntax problems that are quite common when Python is written by a less experienced developer.

Implementing UI

In general, grid computing task expects no direct communication with the owner of the Peer. Interaction may be required at the initialization of the F2F Task by asking the owner permissions to acquire the resources, and at the end of the Task notifying the owner about the release of the resources. Depending on the implementation, these steps can be one-time only and the resource owner may not even know that his resources are being used repeatedly. This is especially the case in the Frid environment where the Frid Peers are "friends". Java-based F2F implementation supports this kind of "friendly" resource acquirement [145]. In the mobile context additional requirements apply to the user interaction (UI) part of the F2F framework. The requirements originate primarily:

- because of the constraints of the mobile device (see Section 2.2), and
- the type of the F2F Job being run.

In the context of "the constraints of the mobile device" the main issues are the limitations of the input and output of a mobile device. The screen for reading the interaction elements may be too small and the keypad to react on the elements too limited (for more detail see 2.2.1). Both of them constraint the communication capabilities with

the user. In the context of "the type of the F2F Job being run", a contradictory with the previous observation arises. A possible application written for the F2F Mobile Computing framework may require more interaction with the user as the application itself may not be of a calculation-only type because it is running in the Mobile context (see Section 2.3).

In the Java-based F2F framework, the extendable SIP Communicator enables writing the UI elements [145]. In the F2F Mobile version Python is used. Porting to targeted mobile operating system Symbian specific UI elements are used (for examples see Chapter 4). Using Symbian UI elements introduced limitations described in Section 3.2.5.

3.2.5 Porting to the Symbian mobile operating system

The F2F Mobile Computing framework is ported to Symbian mobile operating system. The choice was dependent on:

1. the results found while describing the status of the current mobile industry (see Chapter 2), and

2. chosen technology set where F2F Computing core is written in C and interfaced with Python (see Section 3.1).

Mobile devices with a capable hardware set were expected in order to support on-device development and testing. To keep down the mobile networking data costs Wi-Fi [121] support was required for the development purposes. Additionally, the connection over General Packet Radio Service (GPRS) [38] is a must feature for the application to be truly mobile.

Table 3.2 lists and compares the specifications of the mobile devices used in the development and testing of the F2F Mobile Computing framework.

Both devices are technically almost similar having the critical technical requirements to develop F2F Mobile Computing framework:

- Both devices have the same operating system for unified porting capabilities.

- Both devices have WiFi connection for development purposes and GPRS support for live testing.

- Both devices have adequate screen size for user interaction and full QWERTY keyboards for easy input requirements.

In additionally to the real mobile devices, an integral part in development is played by the S60 3rd Edition Software Development Kit (SDK) for Symbian operating system and other tools (see Section 3.4).

F2F Mobile core porting issues

Symbian mobile operating system (OS) version 9.1 for S60 (see table 3.2) is targeted for porting the F2F Mobile Computing framework. The platform is the first major improvement from the previous versions mainly facilitating revised the mandatory (but in many ways contradictory) code signing requirement and changed compiler for the SDK [130] (for more issues on the Symbian see Section 2.2.3). Even with these issues

(the earlier versions have a fatal defect where the phone hangs temporarily after the owner sends hundreds of Short Message Service (SMS) messages [109]) Symbian 9.1 for S60 was released in early 2005.

Symbian operating system is written largely in C++. It supports the development of 3rd party applications. As described in Section 2.2.3 the development task can be complex with even simple programs. Porting an existing C/C++ application to Symbian OS is expected to be even more labor-intensive.

The F2F Mobile core is written in C (see Section 3.1). No direct Symbian specific optimizations are applied to the source. Rewriting the whole code to Symbian specific syntax is a vast task. Open C/C++ for the S60 platform [34] enables porting C/C++ programs to Symbian mobile platform supporting an extensive range of standard C/C++ APIs. The APIs help developers who lack knowledge of the Symbian OS and S60 APIs to contribute applications to Symbian OS. The API contains three main libraries and extensions to be installed onto device:

1. Open C libraries that deliver functions from nine well-known standard POSIX and middleware C libraries: *libc*, *libm*, *libdl* and *libpthread* [35]; *libz* [125]; *libcrypt*, *libcrypto* and *libssl* [77]; and *libglib* [40]. They compose the standard base libraries provided as a vendor-neutral C programming interfaces. Libraries are not implemented fully because of the constraints or limitations of Symbian OS. For example, *fork()* and *exec()* functions are not supported because of the threading model differences in Symbian and UNIX environments [131].

2. Open C++ libraries that incorporate the STLport (a multiplatform C++ Standard Library implementation [107]) including IOStreams, Standard Template Library (STL) and Boost [18] libraries (a collection of free peer-reviewed portable C++ source libraries, that extend the functionality of C++) [132].

3. RGA UI - an extension API for the previous libraries that provide a set of C++ libraries that offer features for the creation of UIs with rich audio and graphics for such applications as games. The API enables users to write feature rich 2D graphical applications by providing access to Device Screen, manipulation of bitmap images, display animated content, draw text, playback audio and video streams [135].

Open C/C++ can be run on all S60 3rd Edition devices. Limitations apply only to RGA UI API. Open C libraries are used to port F2F C source, the F2F Mobile core, onto Symbian operating system.

Jabber.py porting issues

Jabber.py, used in the communication tier of the F2F Mobile Computing framework (see Section 3.2.3), is dependent on two modules - Python's *hashlib* module for hash algorithms and *xml.parsers.expat/pyexpat* module for XML processing.

The *hashlib* module implements a common interface to many different secure hash and message digest algorithms, including the FIPS secure hash algorithms SHA1, SHA224, SHA256, SHA384, and SHA512 as well as RSA MD5 algorithm [134]. The Symbian version of Python does not include the *hashlib* module. In order to support *hashlib* for Jabber.py it has to be ported from a third party development advisable originating from the Python source itself. A backport from Python version 2.5 for use

in versions 2.3 and 2.4 by [161] was used to create a stand alone *hashlib* package for Symbian. Open C API is used to ensure the source to be compatible with Symbian. Since Symbian C/C++ does not support long double, then math-related APIs that use long double might not have the desired precision if to compile at all [131]. Implementation of the SHA384 and SHA512 by [161] are using long doubles. As these hash algorithms are not supported by Symbian and also not used by the Jabber.py, they are excluded from the *hashlib* not affecting the workings of the communication layer. Hashlib is created as Python extended module, has Python layer access for Jabber.py and is included into the F2F Mobile Computing framework package (see Section 3.3).

The *xml.parsers.expat* module is a Python interface to the Expat [30] non-validating XML parser. The module uses the *pyexpat* module to provide access to the Expat parser [134]. Symbian version of the Python does not include *pyexpat* module. In order to support *pyexpat* for Jabber.py it has to be ported from a third party development advisable originating from the Python source itself. A port of *pyexpat* was developed in the Personal Distributed Information Store project at Helsinki Institute for Information Technology [43] to allow the standard Python xml package to be used in programs running on Python for Series 60. It has been tested to run on S60 v2.x (Symbian OS 7.0s, 8.0a, 8.1a) phones, and has been found to compile with Series 60 SDK for Symbian OS versions 1.2 (Symbian OS 6.1) and 2.0 (Symbian OS 7.0) (for more on Symbian OS versions see Section 2.2.3). The distributions contain both the *pyexpat* module port, as well as the unmodified xml module from the Python distribution compatible with the Python version used in S60 devices [88]. The latest version of *pyexpat* port is from November, 2005. It turned out, that the port is also compatible with our targeted Symbian platform 9.1 (even though not listed as the supported Symbian versions) and is suitable for Jabber.py's XML parsing.

I ported the Pyexpat using Open C API and and included it into the F2F Mobile Computing framework package (see Section 3.3).

SWIG porting issues

SWIG (version 1.3.36, release date June 24, 2008) is used to implement Python custom modules (see Section 3.2.2). To build Python extension modules, SWIG uses a layered approach in which parts of the extension module are defined in C and other parts are defined in Python. The layer generates two files:

1. the generated C source file containing the low-level wrappers that need to be compiled and linked with the rest of the F2F Mobile core C application to create an extension module, and

2. the Python source file containing high-level support code that accesses the C layer generated in the first step and presents the layer for Python programs to use.

SWIG relies mostly on the Python/C API (see Section 3.2.2) and uses its macros to generate the correct source depending on the version of Python used. Officially, SWIG does not list Symbian as a supported platform. Python base version 2.2.2 (see next section) is also supported [101]. In the result, when generating SWIG C source for the F2F Mobile core some of the macros referenced are deprecated and have to be manually replaced with the correct versions used in the Symbian Python headers files. These macros are *PyObject_NEW* and *PyObject_DEL*. *PyObject_NEW* is for allocating a

new Python object using the C structure type and *PyObject_DEL* releases the memory allocated to an object using the previous macro [90]. These capitalized versions of the calls are equivalent deprecated versions of the *PyObject_New* and *PyObject_Del* with the same function. In the Symbian version of Python, the capitalized versions of the macros are not present or mapped to the correct ones. They have to be manually changed to the existing lower-case versions.

At the time of writing this thesis, newer versions of SWIG were released (with the latest version 1.3.39, release date March 21, 2009). Main changes in the version updates included Python 3.0 support, method/parameter renaming due to the documentation changes and bug fixes for the Python layer [101]. Expected changes to fix the *PyObject_** references or adding Symbian support in general are not included. As the fixes do not relate directly to the F2F Mobile core requirements, the SWIG version updates where ignored and will be applied as future work (see Section 5).

I used SWIG and Open C API to create the F2F Mobile core as a Python custom module for Symbian mobile operating system (see Section 3.3).

Missing Python modules issues

The Python for S60 Platform (PyS60) simplifies application development and provides a scripting solution for the Symbian C++ APIs. PyS60 version 1.4.5 (latest official update released is November 11, 2008) is based on the Python version 2.2.2 (released in October 14, 2002) [133][90]. The Python for S60 as installed on a S60 device consists of:

- Python runtime package that consists of:
  - Python interpreter.
  - Standard and proprietary Python library modules.
  - S60 UI application framework adaptation component that connects the scripting domain components to the S60 UI.
- Python script shell package that consists of:
  - An application written in Python and visible in the application menu of the device that provides an execution environment for Python scripts.
  - For S60 platform versions prior to 3rd Edition: Python Installer program for installing Python files on the device, which consists of:
    - A recognizer plug-in that recognizes *.py*, *.pyc*, *.pyd* and *.pyo* files as belonging to the Python.
    - Symbian application written in the Python that handles the installation of recognized Python files into the script shell environment.

Python for S60 platform distribution does not include all of the Python standard and optional library modules to save storage space in the phone (for hardware restrictions see Section 2.2). Additionally, some of the modules are extended or are unique because of the Symbian operating system specifics (like user interface, mobile phone specific, contacts, messaging, location and camera modules). These modules are either present in the SDK version of the PyS60 but not installed on the phone, or could be imported

from the Python version 2.2.2. Many of the excluded modules also work in the PyS60 environment without any modifications. F2F Mobile Computing framework relies on some of the modules that are excluded from the PyS60 version 1.4.5. These modules are: *new*, *pickle* and *threading*. All the missing modules are Python source files only and no corresponding C/C++ source of the module need to be ported:

- *new* is used to support the SWIG (see Section 3.2.2) generated Python layer. The *new* module allows an interface to the interpreter object creation functions. This is for use primarily in marshal-type functions, when a new object needs to be created "magically" and not by using the regular creation functions [90].

- *pickle* is used in the communication tier of the F2F Mobile Computing framework. For the specific reasons see Section 3.2.3.

- *threading* is used by the internal logic of the F2F Mobile Computing framework client tier (see Section 3.2.4) for managing F2F Jobs, Tasks and communication (see Section 3.2.3).

The missing modules taken from the Python version 2.2.2 work with the PyS60 version 1.4.5 and are added into the F2F Mobile Computing framework package (see Section 3.3).

At the time of F2F Mobile Computing framework development a newer unofficial version of the PyS60 was released - Python for S60 version 1.9.0 [36] (release date December 24, 2008) starting the 1.9.x series of the PyS60. The release is intended only for S60 3rd Edition and onwards devices. The biggest change in the update, apart form optimizations and device-specific improvements, relies in usage of the Python 2.5.1 core. For the F2F Mobile Computing framework it means supporting all the imported libraries described above including the Expat XML parser for Jabber.py. Open C is also embedded making porting of the existing C programs independent from the additional installation requirement of the Open C/C++ library that is currently done separately. Unfortunately, the releases of the PyS60 version 1.9.x at the time of writing this thesis cannot be used because it is not Symbian signed software. PyS60 version 1.9.x is still considered under development. The F2F Mobile Computing framework is written as a Python extension. The F2F Mobile core itself would rely on the features provided by the underlying Python. It would not require additional capabilities that would require it to be signed (for more on Symbian signed see Section 2.2.3). When the F2F Mobile Computing framework is installed, it writes some of the Python specific module data, in a standard way, into the protected area that cannot be read by Symbian unsigned applications like the newer PyS60 mentioned. Current Symbian signing process does not allow to take advantage of the new PyS60 features. Porting the F2F Mobile Computing framework to a newer PyS60 for lessening its dependencies is planned as a future work (see Section 5).

UI Client issues

As it is described in Section 3.2.4, F2F Peers may not interact with the user in the time of calculating the Task and work on the background. In the mobile context, this may differ (see Section 2.3). The demo application written for the F2F Mobile Computing framework (see Chapter 4) uses user interaction quite extensively throughout the whole application life-cycle. Interaction is a part of the F2F Mobile Computing Task.

PyS60 has a special *appuifw* module that offers an interface to the S60 user interaction (UI) application framework. Concurrency issues apply when using the PyS60 UI module. The thread that initializes the Python interpreter becomes the main Python thread. This is usually the main thread of a UI application. Additionally, a facility called *active object* is used extensively in the Symbian OS to implement co-operative, non-preemptive scheduling within operating system threads (for more on Symbian development constraints see Section 2.2.3). A Python programmer is exposed to these kind of related concurrency issues particularly in UI programming. The standard *thread.lock* cannot normally be used for the synchronization in the UI application main thread, as it blocks the UI event handling that takes place in the same thread context. The Symbian active object based synchronization service called *e32.Ao_lock* can be used to overcome this problem. The main thread can wait in this lock, while the UI remains responsive [133].

In the case of F2F Mobile Computing framework, the main thread is the executing thread. When an F2F Task arrives it is executed as an another thread that may require UI features. F2F Mobile framework will continue its core tasks of F2F management and communications while running the F2F Task. The running F2F Task, as non-main Python thread, is not allowed to manipulate the UI directly - in other words, it cannot use the *appuifw* module directly because it will not be responsive for the user. For the F2F Mobile Task, to access the UI features, F2F Mobile framework as a Python main thread needs to implement and provide the UI elements to the F2F Task. The current version of the F2F Mobile Computing framework provides UI elements for:

- a prompt that can be used for any type of user input, and
- a note for showing short (confirmation) message(s).

When the UI element is being shown, the F2F Mobile Computing framework as the Python main thread, will lock as long as the user has reacted on the UI. This current limitation expects the UI sessions to be as short as possible. For current implementations of the UI see Chapter 4. The future work is applied to improve the current implementation of the UI in F2F Mobile Computing framework (see Chapter 5).

3.3 Compiled package

The compiled F2F Mobile Computing framework package depends on the listed applications installed on the mobile device:

- Python for S60 version 1.4.5 (latest official update release is November 7, 2008) [91].
- Open C/C++ Plug-ins for S60 3rd Edition, edition 1, Open C libraries only (release date May 27, 2008) [34].

For more detail on both see Section 3.2.5.

The F2F Mobile core is compiled into a Symbian application which includes and integrates:

- F2F Mobile core integrated with SWIG [101] (version 1.3.36, release date June 24, 2008) interfaces. For details see Section 3.2.2.

- Jabber.py [51] (version 0.5; release date November 2003). Requirement details see from Section 3.2.3.
- Hashlib module [161] (release date November 19, 2008). For requirement details see Section 3.2.5.
- Pyexpat [88] module (version 1.08, release date November 2, 2005). For requirement details see Section 3.2.5.
- Missing Python modules. For details see Section 3.2.5.

The F2F Mobile Computing framework will be run from the Python interpreter. A main script for starting the framework is added into a required location on the mobile device for Python to access. For the installation process see Chapter 4.

3.4 Development tools

Mobile development is mostly done on the PC platform. Tweaking of the result and the final pre-testing development is done using the real devices. The reasons for this are:

- Deploying a process onto a mobile device can be expensive and slow because of the over-the-air service provisioning[5].
- There are tools for the PC platform that emulate the mobile device and thus speed up the development process.

A set of tools was used for developing the F2F Mobile Computing framework before porting to the targeted devices (see Table 3.2).

Symbian S60 3rd Edition Software Development Kit (SDK)

This SDK allows to develop and implement applications written in C++ for Symbian S60 platform smartphones. Application development with the SDK is PC hosted and includes:

- the needed tools, Application Programming Interfaces (APIs), and documentation for development, and
- an emulator which mimics the functions of a real Symbian S60 smartphone.

The Symbian S60 SDK emulator allows viewing and testing applications on PC before installing them to a real device. It provides a graphical interface of a real phone with the needed phone functionality to test the application. The emulator mimics the operation of an application on a real phone so accurately that application development can be done even before the required hardware, that is, a S60 terminal, is available [130].

Using the emulator the notion of *emulation* has to be taken as is. Application behaviour on the real device may, and generally does, differ compared with the behaviour of the emulator (for more on development constraints on Symbian OS see Section 2.2.3).

[5] mobile application installation process using mobile network communication.

The final application has to be tested on a real device before it is released. Nevertheless, using SDK for mobile software development speeds up the development life cycle and is an integral part of the whole process.

In the development of the F2F Mobile Computing framework major porting work was done using the S60 3rd Edition SDK for Symbian OS, for C++ . The SDK is based on the S60 Developer Platform 3rd Edition and Symbian OS 9.1 and confirms to the targeted devices listed in Table 3.2.

Carbide.c++

Carbide.c++ is a family of integrated development environments (IDEs) for the creation of C++ and C applications for Symbian OS devices. Carbide.c++ is based on the Eclipse IDE [29]. The C++ Development Toolkit, provides the foundation for project and build tools management, as well as the primary interface for the debugger to communicate with the IDE [33]. Pydev [87] plugin for Eclipse introduces Python source code development environment into the Carbide.c++.

Carbide.c++ version 2.0 with integrated S60 3rd Edition SDK and Python/C API was used as the main IDE in the development of the F2F Mobile core custom Python module (for more detail see Section 3.2.2).

OpenFire and Spark

Openfire [47] is a freeware IM and groupchat server that uses the XMPP protocol. Jabber communication protocol is based on the XMPP and Jabber is the communication layer for the F2F Mobile Computation framework (see Section 3.2.3). Spark is an freeware IM client that is supported by Openfire and other Jabber-enabled server.

Openfire version 3.6.2 (release date November 21, 2008) and Spark version 2.5.8 (release data November 14, 2007) where used for developing the F2F Mobile Computing framework communication tier.

3.5 Achievements

Section 1.2 defines a list of the principle requirements envisioned for the F2F Mobile Computing framework. The technological targets were defined to:

- enable the core of F2F Computing - composing Jobs and distributing Tasks for the mobile Peers to compute;
- be lightweight - easy setup, management, usage and extendability;
- enable use of IM communication tools for easy personal and social trust-based F2F communication;
- use P2P techniques fostering fast/direct communication between Peers (like NAT traversal [148] and direct TCP-IP sockets) when the initial IM communication channel underperforms. The framework should always use the best connection available;
- be supported on as many mobile devices as possible.

My work achieved the implementation of the core features of the F2F Mobile Computing framework - the ability to compose, distribute and run the F2F Tasks on mobile Peers (for demo application see Chapter 4).

Job composition is simpler in Python than in Java. Limitations apply to the user interface (UI) possibilities (see 3.2.4) and future work is expected to implement the security solution across the whole framework (see Section 5).

The installation, setup and usability of the F2F Mobile Computing framework has mobile-related quirks (for more see Chapter 2). The current installation process requires installation of the F2F Mobile framework, additional dependency packages and some manual setup (see Section 4.3). Simplifying the installation process is expected in the future work (see Section 5).

The current Instant Messaging (IM) layer uses the Jabber IM protocol. Minimal requirements for inter-Task communication were achieved. Other communication channels are expected in future work (see Section 5).

Applying Peer-to-Peer (P2P) techniques to foster fast/direct communication between the Peers was not achieved. For this, a platform-specific approach could be a solution. This means porting an existing P2P communication library to the Symbian mobile operating system, or writing a custom one. That lies beyond the scope of this thesis and is included as future work (see Section 5).

The decision to base the current F2F Mobile Computing framework on the Symbian mobile operating system was made early on (see Section 3.1). Despite all of the constraints described, support is provided for all Symbian S60 3rd Edition devices. Developing mobile applications generally requires testing on real devices to definitively confirm support (see Sections 3.2.5 and 2.2.3); F2F Mobile frid support was confirmed on two quite different models listed in Table 3.2[6] , which strongly indicates that other same class devices ought to also run the application without problems. This confirms successfully the possibility of achieving F2F Computing on Symbian mobile devices.

[6] Additionally confirmed on Nokia E51, Symbian S60 3rd Ed., Feature pack 1.

		Nokia E61	Nokia E70
General	2G Network	GSM 850 GSM 900 GSM 1800 GSM 1900	GSM 850 (US[2]) GSM 900 (EU[3]) GSM 1800 GSM 1900
	3G Network	UMTS 2100	UMTS 2100 (EU)
	Announced	2005, October	2005, October
	Status	Available	Available
Size	Dimensions	117x69.7x14 mm 108 cc	117x53x22 mm 102 cc
	Weight	144 g	127 g
Display	Type	TFT, 16M colors	TFT, 16M colors
	Size	320x240 pixels 2.9 inches 58x45 mm	352x416 pixels 2.1 inches 35x41 mm
Input	Keypad	Yes	No
	QWERTY	Yes	Yes
	Navigation	5-way navigation	5-way navigation
Memory	Internal	64 MB for RAM 64 MB for storage	64 MB for RAM 64 MB for storage
	Card slot	miniSD, hotswap	miniSD, hotswap
Data	GPRS	Yes	Yes
	HSCSD	No	Yes
	EDGE	Class 10, 236.8 kbps	Class 10, 236.8 kbps
	3G	384 kbps	384 kbps
	WLAN	WiFi 802.11i/e/g VoIP over WLAN	WiFi 802.11i/e/g VoIP over WLAN
	Bluetooth	v1.2	v1.2
	Infrared	Yes	Yes
	USB	Pop-Port	Pop-Port
Features	OS	Symbian OS 9.1[4] Series 60 UI	Symbian OS 9.1 Series 60 UI
	CPU	Dual ARM 9 220 MHz	Dual ARM 9 220 MHz

Table 3.2: Specifications of development devices selected for the F2F Mobile Computing Framework - Nokia E61 and Nokia E70 [72]

Chapter 4

F2F Mobile Computing demo application

The F2F Mobile Computing framework demo application is written as an interactive game. The reason for this is mainly deducted from the outcome described in the Section 2.3. A game is a socially active process and this could be one ideological approach of implementing the F2F Mobile frid Tasks.

This chapter will describe the demo application written for the F2F Mobile Computing framework. Documentation for installation process, setup and execution of the application is provided.

4.1 F2F Mobile Computing application game "What am I thinking about?"

"What am I thinking about?" is a turn-based game implemented using the F2F Mobile Computing framework (see Chapter 3). It will utilize the main achievements made with this thesis (see Section 3.5):

- The game is sent and executed in the mobile Peers as F2F Task using thus the core functionality of the F2F Mobile Computing framework (see Section 3.2.1).
- Jabber-based instant messaging communication is used to set up the Mobile frid network of players; and through F2F the communication will be provided seamlessly to the game Task for in-game communications (for more on communication see Section 3.2.3).

Rules of the game

1. At least three players are required:
   (a) a "thinker", and
   (b) two "guessers".
2. At first, a player will be selected as the "thinker".
3. The "thinker" will bid a word for "guessers" to guess. "What am I thinking about?" is ready to be played.

4. "Thinkers", in round-robin style, start guessing the word:

   (a) by letter, or

   (b) by the whole word.

5. Who guesses the word, will take over the "thinker" role and the game starts all over again.

4.2 Supported devices

The F2F Mobile Computing framework and the game was developed using Symbian S60 3rd Edition devices listed in Table 3.2. Additional testing was done with Nokia E51 [72], also a device of the similar class. Theoretically, all the Symbian S60 3rd Edition devices are supported.

4.3 Installation process

The required software can be acquired from the web page set up by the F2F Mobile Computing framework project [32].

4.3.1 Install and setup the mobile device

The current installation process is described in over-the-air download and installation mode directly from the web site. The alternative way would be to use the appropriate software designed for the over-the-wire application installation. In the case of the mobile devices used in this project (see Table 3.2), the software would be the Nokia PC Suite [73]. The details of using this software will not be described in this thesis.

Install the application environment for the F2F Mobile Computing framework

1. Python application environment (links on the web page: *PyS60 3rd Ed. v1.4.5* and *PyS60 3rd Ed. v1.4.5 Shell*):

   (a) Install the official Python for S60 3rd Edition version 1.4.5 onto the mobile device's main memory - choose *Install to: Phone mem.* and press *Select* when the appropriate prompt is given (refer to the Figure 4.1).

   (b) Wait for the installation process. Installation will take some time and the progress is shown with a generic progress bar similar in the Figure 4.2.

   (c) Confirm the successful installation with a prompt that is generic to every installation process (see Figure 4.3).

   (d) Install the Python for S60 3rd Edition version 1.4.5 Shell with similar steps described previously.

2. Install the Open C/C++ into phone memory in a similar process described with previous points (link on the web page: *Open C*).

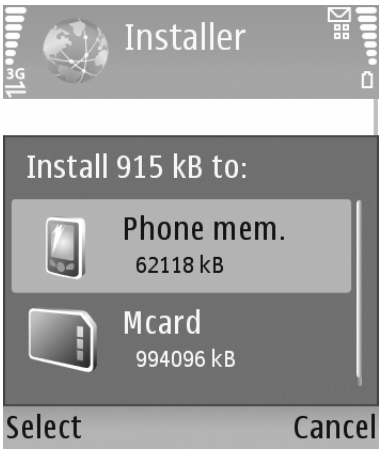

Figure 4.1: Choose the target of the installation (phone memory is the current selection)

3. Install the F2F Mobile Computing framework (link on the web page: *F2F Mobile*):

   (a) Make sure the unsigned applications are allowed to be installed on the device. Under *Application Manager* (*Main menu* -> *Tools* (or *Installations*) -> *App. mgr.*) choose *Options* -> *Settings* and make sure to have:

      i. *Software installation* to *All*

      ii. *Online certif. check* to *Off*

   (b) As the application is not signed a warning message similar to the Figure 4.4 will be shown (for more on Symbian signing requirements see Section 2.2.3). The application can be trusted with selecting *Continue*.

   (c) Continue installing the application into phone memory in a similar process described with Python installation above.

   (d) As the F2F Mobile Framework will be installed as an unsigned Python custom module (see Section 3.2.2) a list of "user-grantable" capabilities are requested (see Figure 4.5) to be accepted by the user (for more on Symbian capabilities see Section 2.2.3). Press *Continue* to allow the capabilities.

   (e) Wait for the package to install and success confirmation to appear.

Setup the mobile client for the F2F Mobile Computing framework

The client will be executing the F2F Mobile Computing framework, initialize the Jabber communication tier, provide it to the F2F Mobile core and start waiting for the F2F Tasks to arrive[1].

1. Make sure the mobile device has suitable and big enough external Secure Digital (SD) memory card. This is required as the current implementation of the client is written as a Python script (see Section 3.2.4), run from the Python shell and searched by Python from the external SD memory card [91, 159].

[1] the game being described in this Chapter

Figure 4.2: Installation is in progress

Figure 4.3: Successful installation confirmed

2. Create a directory, if it is not present yet, named *Python* in the SD memory card. Use the Symbian generic file manager for this task. This directory will be searched by the Python shell.

3. Select one of the F2F Mobile Computing framework client with a set of default settings (descriptive links on the web page: *account pc01 at jabber.cs.ut.ee*, *account pc02 at jabber.cs.ut.ee* and *account pc03 at jabber.cs.ut.ee*) and download it to the created directory in the SD card.

The client is now ready to be run from the mobile device (see Section 4.4) after the PC client is set up properly (see Section 4.3.2).

4.3.2 Install and setup PC

The Microsoft Windows XP-ready PC client is provided. The PC client is required to send out the game as the F2F Task. For the other platforms, the official F2F framework development web site is to be referenced [153, 152]. For PC platform:

1. Install Python [92].

2. Download the ZIP file containing the F2F Computing framework compiled for the Microsoft Windows XP (link on the web page: *F2F framework*).

Figure 4.4: Warning message of an unsigned application

Figure 4.5: "user-grantable" capabilities requested by the unsigned F2F Mobile Computing framework

3. Unpack it to a directory form where the python files inside it can be executed.

The PC client is now ready to be run (see Section 4.4) in conjunction with the installed mobile client (see Section 4.3.1).

4.4 Playing the game

To run the "What am I thinking about?" game at least three players are required (see Section 4.1). One player will be the "thinker" and "guessers" play against each other. As soon as the word is guessed, the winner will be the new "thinker". Additional F2F Peer will be the sender of the game as the F2F Task (see Section 3.2.1). In current implementation, the sender will also be responsible for shutting down the Mobile frid.

To successfully execute and play the game make sure the Section 4.3 has been appropriately followed.

Setup structure of the game

Figure 4.6 describes the setup of the game that will be used as a demo in this Section. It consists of a Mobile frid Peers:

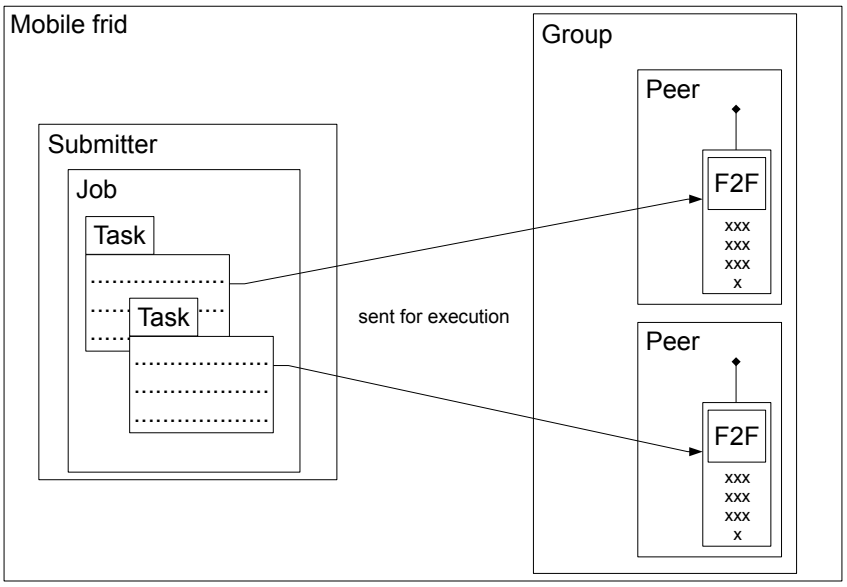

Figure 4.6: Demo setup structure of the game

1. A PC client with Jabber account and referred to as *pc00@jabber.cs.ut.ee* is the "What am I thinking about?" game F2F Task sender. The Peer also controls the lifecycle of the application in the mobile clients.

2. There are three mobile devices that will participate in the game:

   (a) Two mobile phones with Jabber accounts and referred to as *pc01@jabber.cs.ut.ee* and *pc02@jabber.cs.ut.ee*.

   (b) A laptop setup with PC client of the framework. The client has Jabber account and referred to as *pc03@jabber.cs.ut.ee*.

All the Jabber accounts need to be friends with each other. This allows direct messaging (communication) between them and is a prerequisite for the game.

Playing the game

1. Run the F2F Mobile Computing framework clients:

   (a) On the both mobile devices *pc01@jabber.cs.ut.ee.py* and *pc02@jabber.cs.ut.ee.py*:

      i. Start the Python for S60 3rd Edition version 1.4.5 Shell: *Main menu -> Installat. -> Python*.

      ii. Wait for the Shell to start up.

      iii. Run the F2F Mobile Computing framework client (from Python Shell: *Options -> Run script*) *f2f_pc01_jabber.cs.ut.ee.py* (and *f2f_pc02_jabber.cs.ut.ee.py* on the other device). Press *OK*.

      iv. The UI input form will be shown with the loaded default settings for the Jabber account (see Figure 4.7). Make any changes if applicable. Press *Back* to continue.

v. Wait for the mobile device to connect to the network and log into the Jabber account. The process may require choosing the network connection type as a separate prompt. Choose the connection if requested[2]. When the client successfully logs in to the account the *Logged in as pc03 to server jabber.cs.ut.ee* message appears.

(b) On the mobile PC platform *pc03@jabber.cs.ut.ee.py*:

i. Open the command prompt pointing to the directory where the PC client was unzipped (see Section 4.3.2).

ii. Start the F2F Computing framework client using Python (file: *pc03_jabber.cs.ut.ee.py*).

iii. Wait for the confirmation message noting the client is logged into the Jabber server (message appears: *Logged in as pc03 to server jabber.cs.ut.ee*).

2. Send the game to the mobile clients as the F2F Task from the PC client *pc00@jabber.cs.ut.ee.py:*

(a) Open the command prompt pointing to the directory where the PC client was unzipped (see Section 4.3.2).

(b) Start the F2F Computing framework sender client using Python (file: *sender_pc00_jabber.cs.ut.ee.py*).

(c) Wait for the clients to confirm the arrived Task (the game). A UI prompt will be shown on the mobile device to confirm this (see Figure 4.8). Similar textual message will be shown on the command line of the PC client.

(d) When all the mobile clients have received the game send them the initial player data - press any key as *any key for players update...* is being prompted. The *pc00@jabber.cs.ut.ee.py* will then choose the first player to be the "thinker" and the game begins.

3. For playing the game, follow the prompts on mobile devices:

(a) "thinker" is not playing as he's the proposer of the word to guess.

(b) All the players guess the word in the round robin style by:

i. either guessing a letter, or

ii. trying to guess a whole word.

(c) Before the player is asked to guess the currently guessed letters in the word are shown.

(d) If the player guesses the word he/she becomes the new "thinker" proposing the word for others to guess.

4. For shutting down the players, in current implementation, *pc00@jabber.cs.ut.ee.py* needs to notify them:

[2]Using GPRS data may result in high mobile data communication fees. Make sure you know your mobile data plan rates before connecting to the GPRS data networks.

Figure 4.7: Default Jabber account settings

Figure 4.8: F2F Game Task arrived and executed

(a) Press any key to send the quit message to the mobile clients.

(b) Wait for the mobile clients to exit the game.

(c) Exit the Python Shell on the mobile clients:

   i. on the mobile PC this will be automatic.

   ii. on the mobile phones select *Exit*.

Chapter 5

Future work

Section 3.5 lists the targeted achievements for the F2F Mobile Computing framework. The main requirements are achieved but some tasks, either to improve or enhance the current solution, have emerged in the course of the project. These tasks are paved into the future work of the framework described in this chapter.

General networking and communication improvements

Support for other instant messaging clients

The current implementation supports only Jabber instant messaging (IM) client (see Section 3.2.3). The future work is applied to introduce more generic XMPP [122] library (like XMPPPY [123]) in order to support other XMPP-based IM communication mediums like Google Talk [41]. The work could even deprecate the requirement of the current Jabber.py and its own dependencies. As XMPP is a widely used, but still a single-protocol-implementation, additional work is also expected in introducing additional IM clients and protocols to the F2F Mobile Computing framework.

Faster communications

Implementing networking software to the mobile devices can be one of the most complex tasks. Mobile devices can include many different communication mediums (see Section 2.2.1), software specifications can be too abstract having their own set of bugs (see Section 2.2.2) and the operators can apply hidden *walled gardens* to limit the networking capabilities (see Section 2.2.4). The generic approach of implementing networking software on the mobile devices should follow a KISS principle [57]. F2F Computing requires definitely more advanced communications than currently proof-of-concept adequate, but relatively slow, Jabber-based IM communication layer (see Section 3.2.3). Future work is applied in research and implementation of either generic or mobile platform specific fast communication libraries that would foster possible Peer-to-Peer techniques between highly mobile F2F Peers. IM communication can only be the initiator of the F2F Mobile Computing transaction.

Reducing dependencies

Updating PyS60

The current implementation of the F2F Mobile Computing framework is limited to Symbian S60 3rd Edition mobile operating system, Open C/C++ for the S60, Python for S60 (PyS60) and the missing Python libraries (see Section 3.2.5). Newer version of the Python 1.9.x [36] will include the missing libraries that are currently manually added to the F2F Mobile Computing framework. This would include XML parsing libraries for XMPP-based communications, Open C/C++ for the S60 and other Python modules missing from the official version of PyS60 version 1.4.5. Current F2F Mobile Computing framework implementation requires the Python to be signed (for more on Symbian signing see Section 2.2.3). As PyS60 version 1.9.x is considered under the development and is not signed, the future work is applied in reducing the dependencies by applying the newer Python for S60 as soon as it will be officially released.

Updating SWIG

SWIG is used to implement the F2F Mobile Computing framework as Python custom module (see Section 3.2.2). SWIG is not officially supported by the Symbian mobile operating system. With some manual rework the SWIG generated Python/C API is being used. The future work is applied in possible replacement of the SWIG with either some alternative (like Pyrex [89]and Cython [25]) or manually created optimally designed Python custom module wrapper for the F2F Mobile Computing framework.

Usability

From the usability side, the process of installation and setup should have lesser steps than it currently has (see Section 4.3). The target is to have one installable package. The mobile clients should not install multiple packages that is rather uncomfortable process on a constrained mobile device and can become an expensive for the user (see Sections 2.2.1 and 2.2.4). The work is in direct correlation with work applied to the reducing of the dependencies.

Enhanced user interaction capabilities

Current F2F Mobile Computing framework user interaction (UI) capabilities are limited to two UI elements (see Section 3.2.5) that are required by the demo application (see 4). The limitation is partly due to the technology selection and partly due to the implementation of the client tier (see Section 3.2.4). The future work is applied in rework of the current UI library support for the F2F Mobile Computing framework and is related to the work concerning reducing the dependencies where new updates to the current set of technologies may introduce new tools that can help to solve the current issue.

Mobile frid Group management

Current F2F Mobile Computing framework implementation manages the Mobile frid friends and Groups (see Section 3.2.1) manually. This means, that there is no elegant way of adding, removing or managing the friends in the Mobile frid Group other than the "default setup" without any confirmation from the Peers. The future work is applied introducing more elegant and dynamic way of setting up the Mobile frid Groups as a part of the running program itself. This would include the Peer invitation(s) (either manual or half-automated with possible Public Key Infrastructure trust model), management, Task coordination and other Group management tasks. The work is in direct relationship with enhancing the user interface features of the current implementation.

Porting to other mobile platforms

Current implementation is tested and ported to Symbian S60 3rd Edition mobile operating system. The future work is applied to support other mobile operating systems. The work is not expected to be simple and have to be considered with caution taking into the consideration the fast mobile industry change characteristics (see Section 2.1.3) and the complexity and fragmentation of the mobile operating systems (see Section 2.2.2). From research point of view the more innovative platforms like Google Android [7] and Linux-based Limo [60] mobile devices could be an interesting choices to boost their popularity in the Mobile Computing research.

Security

Current implementation does not implement any security to the F2F Mobile Computing framework. For example, current implementation does not verify the integrity and reliability of the Task received as a plain executable Python source code (see Section 3.2.2). The framework will execute any Task it will receive. The future work is applied to the research and implementation of the security layer that will be a part of the F2F Mobile Computing framework. The work has to consider the limitations found the mobile industry (see Chapter 2) and the other possible future work applied in the areas of communications, user interaction and reducing dependencies.

Summary

The main task of this thesis is a feasibility study to use Friend-to-Friend (F2F) Computing in a mobile environment. F2F Computing was initiated in Spring 2007 by a group of students, including myself, and conducted by Ulrich Norbisrath [154] from the Distributed Systems Group [27] at the University of Tartu[112]. F2F Computing is a simple concept of distributed computing where participants are each other's friends and where the computational tasks can be shared with each other as easily as friendship. My task was to port the ideology and existing F2F Computing functionality on to mobile phones. This led to a proof of concept for F2F's ability to run inside an environment that is highly mobile, generally very restricted and potentially limited in resources. The work additionally provides a template for how F2F Computing applications should be implemented across a Mobile Grid framework.

Before beginning the F2F Mobile Computing framework implementation, an understanding of the mobile phone and mobile industry is required. The whole industry is only a couple of decades old, and has only been mainstream for less than one decade. In spite of the scarce formal information available, I managed to compose and describe a short history of the mobile phone, mobile networks and the current status of the mobile industry through the eyes of a mobile software developer.

In recent years the mobile phone has transformed from a consumer electronics black box to a programmable platform, which has drawn in increasing numbers of 3rd party developers to exploit this potential. One of the biggest challenges for any mobile software developer, mobile fragmentation, is described following the mobile industry summary. Mobile fragmentation leads to a situation where a mobile software developer must manage a huge number of different mobile phones with different hardware specifications, software platforms, and external parties like operators and other 3rd party mobile software providers. The existence of this platform fragmentation, alongside the inherent restrictions of small portable devices, impose constraints on the mobile software developer. In this thesis I describe the existence of the three main classes of constraint that directly affect mobile software development: hardware, software and operator constraints. All the descriptions include examples from the real world and possible solutions and/or workarounds for the issues that were expected when developing the F2F Mobile Computing framework.

After understanding the state and complexity of the mobile industry, development of the F2F Mobile Computing was ready to begin. For that, the most suitable mobile operating system is determined to be Symbian. It currently accounts for more than 40% of the smartphones in the market, with around 250 different mobile phone models running the Symbian mobile operating system. It supports 3rd party software development with a rich array of languages and tools, including the core set of technologies that the F2F Mobile Computing framework requires.

The F2F Computing framework, basis for the F2F Mobile Computing framework,

was implemented by Ulrich Norbisrath in C and Python [154]. I ported this version of the framework to the Symbian mobile operating system, expecting the constraints found in the mobile software development. This thesis describes the work done in the process of porting which includes using additional technologies like the Python application environment for Symbian, Open C/C++ for Symbian, and other tools and missing libraries to overcome constraints found in Symbian. The implementation contains the main features of F2F Mobile Computing - composition, distribution and execution of F2F Mobile Grid tasks (also known as Mobile frid tasks) within the mobile environment. The feature creates a Mobile Grid computing infrastructure using the F2F Computing ideology and approach. The current implementation has theoretical support for all of the Symbian S60 3rd Edition devices, and was tested and confirmed on a selected subset of these smartphones.

I created a demo application for testing and confirming the F2F Mobile Computing framework features. It is an interactive game distributed as a Mobile frid task to the mobile clients running the F2F Mobile Computing framework. The whole setup process for the required libraries and F2F Mobile Computing framework is provided.

In the process of writing this thesis I wanted to find answers to a couple of general questions, such as:

1. Can mobile phones be used as grid computing devices?

2. What are the main obstacles in supporting grid-like frameworks on mobile devices?

The first question was answered with a resounding yes - mobile phones can be used as a grid computing devices. The potential of billions of connected mobile phones is alluring as a computational resource. As a single isolated device, they can achieve little but viewed as a hive of devices, the industry has potential. Problems still exist in implementing a working and usable model for distributed mobile computing. One of the questions is: how should a mobile distributed application be written? As it turned out, a mobile device is a very personal social tool. This means that the applications in it cannot just run as a "black computational box". The Friend-to-Friend Computing framework's ideological approach is social-friendly, which is highly complementary to the nature of a mobile device and seems essential to the success of mobile distributed computing. The other problems are technical. These have historically arisen from the evolution of the mobile industry, which is currently fragmented with a complex set of different technologies. As a result, software for mobile phones must be written in a fragmented manner, balancing development and maintenance complexity with target market size, and in some cases ends up targeted at only a single mobile platform or device. There are numerous initiatives to solve mobile fragmentation but arguably fragmentation is worse today than at any point in the past, which makes creating a generic implementation of a complex platform like F2F Mobile Computing a challenge. Nevertheless, this implementation of the F2F Mobile Computing framework shows clearly that Mobile Grid applications are possible.

Mobiilistuv sõprusraalimine

Magistritöö (20 AP)

Sven Kirsimäe

Sisukokkuvõte

Antud magistriöö eesmärgiks oli uurida võimalikku Tartu Ülikooli Matemaatika- Informaatikateaduskonnas 2007. aastal algatatud sõprusraalimise (*Friend-to-Friend Computing*) ideoloogia ja funktsionaalsuse rakendamist mobiiltelefonidel.

Töö autor osales ise sõprusraalimise algatusgrupis, mida juhendas Hajussüsteemide grupi liige ja õppejõud Ulrich Norbisrath. Algatusgrupi töö tulemusena sündis sõprusraalimise idee ja esmane teostus. Sellest ajast alates on antud valdkonnas tehtud oluliselt palju uurimistööd [153]. Samal ajal töö autor professionaalsel tasemel tarkvara arendusega mobiilvaldkonnas. Peagi tekkis idee rakendada saadud kogemust sõprusraalimise uurimisvaldkonnas - tekkis idee luua sõprusraalimine mobiiltelefonidel. Uurimus annaks aimu antud raamistiku rakendamise võimalikkusest tehnilises keskkonnas, mis on ülimalt liikuv, üldiselt tõkestatud ning piiratud ressurssidega. Töö annab vastuse sõprusraalimise sobilikkusest mobiilses võrkraalimises (*Mobile grid*) ning võimalikke lahendusi selle efektiivseks saavutmiseks.

Esmalt kirjeldatakse töös mobiilvaldkonna hetkeseisu ja keerukust. Kuigi maailmas on miljardeid aktiivseid mobiiltelefonide kasutajaid, on mobiilvaldkonna hetkeseisu kuvandi leidmine keerukas ülesanne. Kogu valdkond on paarikümne aasta vanune. Muutused valdkonnas on kiired. Uuenduslikud mobiiltehnoloogiad võetakse kiiresti kasutusele ning on valdkonnas peagi üldlevinud. Arusaam mobiilvaldkonnast on tähtis just mobiiltarkvara loome aspektist, sest mobiilne sõprusraalimine on tarkvaraline rakendus mobiiltelefonidele. Magistritöös kirjeldatakse mobiiltelefonide ja -võrkude tekkelugu, ning hetkeseisu mobiiltarkvara arendaja silme läbi. Järgnevalt lahatakse mobiiltarkvara loomise keerukust, mis tuleneb mobiilvaldkonna killustatusest (*fragmentation*). Mobiilvaldkonna killustatus on seisund, kus peab arvestama mobiiltelefonide riistvaraliste erisustega, mitmete tarkvaraliste lahendustega mobiiltelefonide käitamiseks ning teiste väliste mõjutajatega nagu näites mobiilvõrkude operaatorite poolt rakendatavad kitsendused. Töö käsitleb riist-, tarkvara ning mobiiloperaatorite poolt tekitatud piiratust mobiilvaldkonnas. Töö sisaldab näited elust enesest ning võimalikke lahendusi piirangute vältimiseks. Kõikide kirjeldatud piirangute olemasolu on arvestatud mobiilse sõprusraalimise raamistiku (*F2F Mobile Computing framework*) loomisel.

Mobiilvaldkonna hetkeseisu uurimisel jõuti järelduseni võimaliku mobiilplatvormi valikuks mobiilse sõprusraalimise platvormi realiseerimiseks. Selleks osutus *Symbian* mobiilplatvorm. Tegemist on ühe populaarseima mobiiltelefonide operatsioonisüsteemiga tänapäeval. Magistritöö praktilise osana kirjeldatakse tehnilisi samme ning lisatehnoloogiaid, mis osutusid vajalikuks sõprusraalimise realiseerimiseks *Symbian* mobiilplatvormile.

Töö rakendatavuse eesmärgiks sai loodud näidisprogramm. Näidisprogramm rakendab mobiilse sõprusraalimise põhilist funktsionaalsust, milleks on võrkraalimise võimalikkus mobiilplatvormil kasutades sõprusraalimise ideoloogiat ja lähenemist.

Töö tulemusena täideti peamine püstitatud eesmärk kinnitamaks sõprusraalimise võimalikkust tehniliselt ülimalt liikuvas, tõkestatud ning piiratud keskkonnas. Näidisrakendus tõestab, et sõprusraalimine mobiilses võrkraalimises on võimalik.

Bibliography

[1] AdMob - about. http://www.admob.com/home/about [Last accessed on April 20, 2009].

[2] Adobe - flash lite. http://www.adobe.com/products/flashlite/ [Last accessed on May 24, 2009].

[3] Adobe - mobile and digital home. http://www.adobe.com/mobile/ [Last accessed on May 24, 2009].

[4] Adobe flash player. http://www.adobe.com/products/flashplayer/ [Last accessed on May 24, 2009].

[5] Akogrimo. http://www.akogrimo.org/ [Last accessed on May 24, 2009].

[6] Android | market. http://www.android.com/market/ [Last accessed on May 24, 2009].

[7] Android | official website. http://www.android.com/ [Last accessed on May 24, 2009].

[8] AOL.com - welcome to AOL. http://www.aol.com/ [Last accessed on May 24, 2009].

[9] Apple - iPhone. http://www.apple.com/iphone/ [Last accessed on May 24, 2009].

[10] Apple - iPhone - app store and applications for iPhone. http://www.apple.com/iphone/appstore/ [Last accessed on May 24, 2009].

[11] Apple - iPhone - technical specifications. http://www.apple.com/iphone/specs.html [Last accessed on May 24, 2009].

[12] Apple - mac OS x leopard. http://www.apple.com/macosx/ [Last accessed on May 24, 2009].

[13] Apple inc. http://www.apple.com/ [Last accessed on May 24, 2009].

[14] ARM information center. http://infocenter.arm.com/help/index.jsp [Last accessed on April 26, 2009].

[15] AT&T devCentral | java signing specification. http://developer.att.com/developer/index.jsp?page=toolsTechDetail& id=11300207 [Last accessed on April 25, 2009].

[16] Bell labs. http://www.bell-labs.com/ [Last accessed on May 24, 2009].

[17] BlackBerry - app world – the official BlackBerry app store for BlackBerry apps. http://na.blackberry.com/eng/services/appworld/ [Last accessed on May 24, 2009].

[18] Boost c++ libraries. http://www.boost.org/ [Last accessed on May 24, 2009].

[19] C (programming language). http://en.wikipedia.org/wiki/C_(programming_language) [Last accessed on March 20, 2009].

[20] Cell phones, mobile phones, and wireless calling plans from sprint. http://www.sprint.com/ [Last accessed on May 24, 2009].

[21] Cell Phones,Cellular phone Plans,Prepaid cell Phones,Free cell phones & deals - stick together with T-Mobile. http://www.t-mobile.com/ [Last accessed on May 24, 2009].

[22] Cellular network - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Cellular_network [Last accessed on May 24, 2009].

[23] Common lisp. http://common-lisp.net/ [Last accessed on May 24, 2009].

[24] comScore, inc. - mobile products and services. http://www.mmetrics.com/ [Last accessed on May 24, 2009].

[25] Cython: C-Extensions for python. http://www.cython.org/ [Last accessed on March 21, 2009].

[26] Digital rights management - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Digital_rights_management [Last accessed on May 24, 2009].

[27] Distributed systems group site. http://ds.cs.ut.ee/ [Last accessed on March 18, 2009].

[28] Eclipse public license - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Eclipse_Public_License [Last accessed on May 24, 2009].

[29] Eclipse.org. http://www.eclipse.org/ [Last accessed on May 24, 2009].

[30] The expat XML parser. http://www.libexpat.org/ [Last accessed on May 24, 2009].

[31] Extensible messaging and presence protocol - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol [Last accessed on March 22, 2009].

[32] F2F mobile computing (The frid goes mobile). http://ulno.net/f2f/mobile [Last accessed on May 24, 2009].

[33] Forum nokia - carbide.c++. http://www.forum.nokia.com/Resources_and_ Information/Tools/IDEs/Carbide.c++/ [Last accessed on April 8, 2009].

[34] Forum nokia - open C/C++. http://www.forum.nokia.com/Resources_and_Information/Explore/Runtime_Platforms/Open_C_and_C++/ [Last accessed on March 31, 2009].

[35] The FreeBSD project. http://www.freebsd.org/ [Last accessed on May 24, 2009].

[36] garage: Python for s60: Project filelist. https://garage.maemo.org/frs/?group_id=854 [Last accessed on May 24, 2009].

[37] Gartner consulting. http://www.gartner.com/ [Last accessed on May 24, 2009].

[38] General packet radio service - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/GPRS [Last accessed on May 24, 2009].

[39] GetJar: free java, symbian, windows mobile, BlackBerry, palm and flash lite mobile games and applications. http://www.getjar.com/site/info [Last accessed on April 20, 2009].

[40] GNOME: the free software desktop project. http://www.gnome.org/ [Last accessed on May 24, 2009].

[41] Google talk - chat online and make free internet calls. http://www.google.com/talk/ [Last accessed on May 24, 2009].

[42] Group prods FCC to defend skype on iPhone - WSJ.com. http://online.wsj.com/article/SB123876873806886721.html [Last accessed on April 24, 2009].

[43] Helsinki institute for information technology HIIT | HIIT. http://www.hiit.fi/ [Last accessed on May 24, 2009].

[44] High-performance computing - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/High-performance_computing [Last accessed on March 25, 2009].

[45] History of mobile phones - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/History_of_mobile_phones [Last accessed on April 19, 2009].

[46] ICQ.com - community, people search and messaging service! http://www.icq.com/ [Last accessed on May 24, 2009].

[47] Ignite realtime: Openfire server. http://www.igniterealtime.org/projects/openfire/ [Last accessed on May 24, 2009].

[48] iPhone app and iPhone game reviews and news :: 148Apps. http://www.148apps.com/ [Last accessed on May 24, 2009].

[49] J2ME polish. http://www.j2mepolish.org/ [Last accessed on April 21, 2009].

[50] jabber.org. http://www.jabber.org/ [Last accessed on March 22, 2009].

[51] jabber.py - a python jabber library. http://jabberpy.sourceforge.net/ [Last accessed on March 22, 2009].

[52] The java community Process(SM) program - JSRs: java specification requests - detail JSR# 248 (MSA). http://jcp.org/en/jsr/detail?id=248 [Last accessed on May 24, 2009].

[53] Java (programming language). http://en.wikipedia.org/wiki/Java_(programming_language) [Last accessed on March 18, 2009].

[54] Java verified program home. http://www.javaverified.com/ [Last accessed on May 24, 2009].

[55] java.com: Java + you. http://www.java.com/ [Last accessed on May 24, 2009].

[56] K*Grid mobile grid. http://www.gridcenter.or.kr/MobileGrid/ [Last accessed on May 24, 2009].

[57] KISS principle - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/KISS_principle [Last accessed on May 24, 2009].

[58] Laptop, notebook, desktop, server and embedded processor technology - intel. http://www.intel.com/ [Last accessed on May 24, 2009].

[59] LG mobile. http://www.lgmobile.com/ [Last accessed on May 24, 2009].

[60] LiMo foundation. http://www.limofoundation.org/ [Last accessed on May 24, 2009].

[61] Making history: Developing the portable cellular system. http://www.motorola.com/content.jsp?globalObjectId=7662-10813 [Last accessed on May 24, 2009].

[62] *Making search social - Unleashing search for the mobile generation.* Taptu.

[63] *Measuring the Information Society - The ICT Development Index.* International Telecommunication Union.

[64] Microsoft corporation. http://www.microsoft.com/ [Last accessed on May 24, 2009].

[65] Microsoft windows mobile. http://www.microsoft.com/windowsmobile/ [Last accessed on May 24, 2009].

[66] M:Metrics news - social networking and commerce draw consumers into the mobile web. http://www.mmetrics.com/press/PressRelease.aspx?article=20080521-smartbrowsing [Last accessed on April 12, 2009].

[67] Mobile network operator - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Mobile_network_operator [Last accessed on April 24, 2009].

[68] Mobile operating system - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Mobile_platform [Last accessed on April 22, 2009].

[69] Modula-3 resource page. http://www.modula3.org/ [Last accessed on May 24, 2009].

[70] Motorola. http://www.motorola.com/ [Last accessed on May 24, 2009].

[71] MSN.com. http://www.msn.com/ [Last accessed on May 24, 2009].

[72] Nokia - nokia on the web. http://www.nokia.com/ [Last accessed on March 29, 2009].

[73] Nokia europe - nokia PC suite. http://europe.nokia.com/A4144903 [Last accessed on May 24, 2009].

[74] NTT DOCOMO. http://www.nttdocomo.com/ [Last accessed on May 24, 2009].

[75] Nucleus RTOS. http://www.mentor.com/products/embedded_software/nucleus_rtos/index.cfm [Last accessed on May 24, 2009].

[76] Octave. http://www.gnu.org/software/octave/ [Last accessed on May 24, 2009].

[77] OpenSSL: the open source toolkit for SSL/TLS. http://www.openssl.org/ [Last accessed on May 24, 2009].

[78] Orange. http://www.orange.com/ [Last accessed on May 24, 2009].

[79] Orange and vodafone cut VoIP from nokia n95 - iTnews australia.

[80] Orange application shop. http://orangestore.cellmania.com/ [Last accessed on May 24, 2009].

[81] OSE Real-Time operating systems (RTOS) - enea. http://www.enea.com/ose [Last accessed on May 24, 2009].

[82] Ovi share by nokia - share your photos and videos. anytime. anywhere. http://share.ovi.com/ [Last accessed on May 24, 2009].

[83] Palm - software store. http://software.palm.com/us/html/ [Last accessed on May 24, 2009].

[84] The perl directory - perl.org. http://www.perl.org/ [Last accessed on May 24, 2009].

[85] PHP: hypertext preprocessor. http://php.net/ [Last accessed on May 24, 2009].

[86] The programming language lua. http://www.lua.org/ [Last accessed on May 24, 2009].

[87] Pydev. http://pydev.sourceforge.net/ [Last accessed on May 24, 2009].

[88] pyexpat for python series 60. http://pdis.hiit.fi/pdis/download/pyexpat/ [Last accessed on March 31, 2009].

[89] Pyrex. http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/ [Last accessed on March 21, 2009].

[90] Python 2.2.2 documentation. http://www.python.org/doc/2.2.2/ [Last accessed on April 1, 2009].

[91] Python for s60. http://opensource.nokia.com/projects/pythonfors60/ [Last accessed on April 5, 2009].

[92] Python programming language – official website. http://www.python.org/ [Last accessed on March 20, 2009].

[93] Python (programming language) - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Python_(programming_language) [Last accessed on March 21, 2009].

[94] Qualcomm home. http://www.qualcomm.com/ [Last accessed on May 24, 2009].

[95] Real-time operating system - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Real-time_operating_system [Last accessed on April 22, 2009].

[96] Research in motion. http://www.rim.com/ [Last accessed on May 24, 2009].

[97] Ruby programming language. http://www.ruby-lang.org/ [Last accessed on May 24, 2009].

[98] Samsung. http://www.samsung.com/ [Last accessed on May 24, 2009].

[99] The scheme programming language. http://groups.csail.mit.edu/mac/projects/scheme/ [Last accessed on May 24, 2009].

[100] SIM lock - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/SIM_lock [Last accessed on May 24, 2009].

[101] Simplified wrapper and interface generator. http://www.swig.org/ [Last accessed on March 21, 2009].

[102] SIP communicator. http://sip-communicator.org/ [Last accessed on March 17, 2009].

[103] Skype. http://skype.com/intl/en-gb/welcomeback/ [Last accessed on March 17, 2009].

[104] Skype4Java. https://developer.skype.com/wiki/Java_API [Last accessed on March 17, 2009].

[105] Sony ericsson. http://www.sonyericsson.com/cws/home?cc=ee&lc=et [Last accessed on May 24, 2009].

[106] StatCounter free invisible web tracker, hit counter and web stats. http://www.statcounter.com/, http://gs.statcounter.com [Last accessed on April 20, 2009].

[107] STLport. http://www.stlport.org/ [Last accessed on May 24, 2009].

[108] SureType - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/SureType [Last accessed on May 24, 2009].

[109] Symbian OS. http://en.wikipedia.org/wiki/Symbian_OS [Last accessed on March 16, 2009].

[110] Symbian signed. https://www.symbiansigned.com/ [Last accessed on May 24, 2009].

[111] Symbian signed test houses. https://www.symbiansigned.com/app/page/overview/testhouses [Last accessed on May 24, 2009].

[112] Tartu Ülikool. http://www.ut.ee/ [Last accessed on May 24, 2009].

[113] Tcl developer site. http://www.tcl.tk/ [Last accessed on May 24, 2009].

[114] Telephone exchange names - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Telephone_exchange_names [Last accessed on May 24, 2009].

[115] Toshiba mobile | toshiba TG01. https://www.toshiba-europe.com/mobilerevolution/ [Last accessed on May 24, 2009].

[116] Trackball - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Trackball [Last accessed on May 24, 2009].

[117] Verizon. http://www22.verizon.com/ [Last accessed on May 24, 2009].

[118] VisionMobile :: market analysis and strategic advisory. http://www.visionmobile.com/ [Last accessed on May 24, 2009].

[119] Vodafone. http://www.vodafone.com/ [Last accessed on May 24, 2009].

[120] WAP gateway - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/WAP_gateway [Last accessed on May 24, 2009].

[121] Wi-Fi - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Wi-Fi [Last accessed on March 29, 2009].

[122] XMPP standards foundation. http://xmpp.org/ [Last accessed on March 22, 2009].

[123] xmpppy: the jabber python project. http://xmpppy.sourceforge.net/ [Last accessed on March 22, 2009].

[124] Yahoo! http://www.yahoo.com/ [Last accessed on May 24, 2009].

[125] zlib home site. http://www.zlib.net/ [Last accessed on May 24, 2009].

[126] *Connected, Limited Device Configuration.* Specification Version 1.0. Sun Microsystems, Inc., May 2000.

[127] *Mobile Information Device Profile (JSR-37).* Java 2 Platform, Micro Edition, 1.0. Sun Microsystems, Inc., December 2000.

[128] *Mobile Information Device Profile.* Version 2.0. Sun Microsystems, Inc., November 2002.

[129] *Connected, Limited Device Configuration.* Specification Version 1.1. Sun Microsystems, Inc., March 2003.

[130] *S60 3rd Edition SDK for Symbian OS.* Nokia, 2006.

[131] *Open C for S60 3rd Edition SDK for Symbian OS.* Nokia, 2008.

[132] *Open C++ for S60 3rd Edition SDK for Symbian OS.* Nokia, 2008.

[133] *PyS60 Library Reference.* Release 1.4.5

final edition, December 2008.

[134] *Python v2.6 documentation.* Python Software Foundation, October 2008.

[135] *Realtime Graphics and Audio APIs (RGA) for S60 3rd Edition SDK for Symbian OS, for C++.* Nokia, 2008.

[136] *AdMob Mobile Metrics Report.* AdMob, February 2009.

[137] *Netsize Publishes Netsize Guide 2009: Mobile Society & Me, When Worlds Combine.* Netsize, 2009.

[138] Jon Agar. *Constant Touch: A Global History of the Mobile Phone.* Totem Books, February 2005.

[139] Sanjay P. Ahuja and Jack R. Myers. A survey on wireless grid computing. *J. Supercomput.*, 37(1):3–21.

[140] F. Berman, G. Fox, and A. J. G. Hey. *Grid computing: making the global infrastructure a reality.* Wiley, 2003.

[141] D. Bruneo, M. Scarpa, A. Zaia, and A. Puliafito. Communication paradigms for mobile grid users. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 669–676, 2003.

[142] Leigh Edwards, Richard Barker, and Staff of EMCC Software Ltd. *Developing Series 60 Applications: A Guide for Symbian OS C++ Developers.* Addison-Wesley Professional, March 2004.

[143] Peter Glotz and Stefan Bertsch. *Thumb Culture: The Meaning of Mobile Phones for Society.* Transcript Verlag, illustrated edition edition, October 2005.

[144] Oleg Knut. A low level virtual machine backend for F2F computing. http://ulno.net/f2f/llvm [Last accessed on March 18, 2009].

[145] K. Kraaner. *Friend-to-Friend Computing Instant Messaging Based Spontaneous Desktop Grid.* Master thesis, University of Tartu.

[146] Liz Laffan, Andreas Constantinou, and VisionMobile. Mobile software management. advances and opportunities in service delivery. http://www.visionmobile.com/research.php#msm [Last accessed on April 12, 2009].

[147] Sing Li and Jonathan Knudsen. *Beginning J2ME: From Novice to Professional, Third Edition.* Apress, 3rd edition, April 2005.

[148] Artjom Lind. *NAT Traversal in P2P systems in Java.* Bachelor thesis, University of Tartu, June 2008.

[149] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30.

[150] Dmitiri Melnikov. F2F computing as a base for network games - bub's brothers. http://ulno.net/f2f/interactive/bbros [Last accessed on March 18, 2009].

[151] F. Navarro, A. Schulter, F. Koch, M. Assuncao, and C.B. Westphall. Towards a middleware for mobile grids. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 1–4, 2006.

[152] Ulrich Norbisrath. F2F computing development - getting started. http://ulno.net/f2f/development [Last accessed on March 18, 2009].

[153] Ulrich Norbisrath. Friend-to-Friend (F2F) computing. http://ulno.net/f2f [Last accessed on March 17, 2009].

[154] Ulrich Norbisrath. Teaching. http://ulno.net/teaching/sp#2007Spring [Last accessed on March 17, 2009].

[155] Luca Passani. WURFL. http://wurfl.sourceforge.net/ [Last accessed on April 21, 2009].

[156] Indrek Priks. *F2F network topology visualization.* Master thesis, University of Tartu.

[157] V. Raghunathan, T. Pering, R. Want, A. Nguyen, and P. Jensen. Experience with a low power wireless mobile computing platform. In *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, pages 363–368, 2004.

[158] M. Satyanarayanan. Mobile computing. *Computer*, 26(9):81–82, 1993.

[159] Jürgen Scheible and Ville Tuulos. *Mobile Python: Rapid prototyping of applications on the mobile platform.* Wiley, December 2007.

[160] Mark Shackman. *Platform Security - a Technical Overview.* Version 1.2. Nokia.

[161] Gregory Smith. [code.krypto.org : python/hashlib]. http://code.krypto.org/python/hashlib/ [Last accessed on March 31, 2009].

[162] Guido van Rossum. *Extending and Embedding Python.* Release 2.6.1. March 2009.

[163] Guido van Rossum. *The Python Language Reference*. Release 2.6.1. March 2009.

[164] VisionMobile. The 100 million club. http://www.visionmobile.com/rsc/researchreports/VisionMobile-100-million-club-(1H08).pdf [Last accessed on April 16, 2009].

[165] VisionMobile. Mobile industry atlas (Wallchart). http://www.visionmobile.com/research.php [Last accessed on April 16, 2009].

[166] VisionMobile. Mobile megatrends 2009. http://www.visionmobile.com/blog/2009/02/mobile-megatrends-2009/ [Last accessed on April 16, 2009].