VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

Domantė Olekaitė

MOBILIOJO PIRKĖJO GIDO ĮGYVENDINIMAS IR OPTIMIZACIJOS TYRIMAS: MOBILIOJI PLATFORMA

MOBILE SHOPPER GUIDE: IMPLEMENTATION AND OPTIMIZATION ANALYSIS OF THE MOBILE PLATFORM

Baigiamasis magistro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 62407T104

Informacinių technologijų specializacija

Duomenų gavybos mokslo kryptis

Vilnius, 2008

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

TVIRTINU
Katedros vedėjas

_____
(Parašas)
Prof. habil. dr.Genadijus Kulvietis
(Vardas, pavardė)

_____
(Data)

Domantė Olekaitė

MOBILIOJO PIRKĖJO GIDO ĮGYVENDINIMAS IR OPTIMIZACIJOS TYRIMAS: MOBILIOJI PLATFORMA

MOBILE SHOPPER GUIDE: IMPLEMENTATION AND OPTIMIZATION ANALYSIS OF THE MOBILE PLATFORM

Baigiamasis magistro darbas

Informacinių technologijų studijų programa, valstybinis kodas 62407T104

Duomenų gavybos specializacija

Inžinerinės informatikos mokslo kryptis

Vadovas doc. dr. A. Čepulkauskas _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Konsultantas_____ _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Konsultantas_____ _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Vilnius, 2008

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Informacinių technologijų katedra

ISBN ISSN
Egz. sk. …2…..
Data ….-….-….

Duomenų gavybos studijų programos baigiamasis magistro darbas

Pavadinimas **Mobiliojo pirkėjo gido įgyvendinimas ir optimizacijos tyrimas: mobilioji platforma**

Autorius **Domantė Olekaitė** Vadovas doc. dr. **Algimantas Čepulkauskas**

Kalba

[ ] lietuvių

[X] anglų

Anotacija

Baigiamajame magistro darbe įgyvendinama *Mobiliojo pirkėjo gido* taikomoji programa mobiliajame telefone, pateikianti miesto žemėlapį su parduotuvėmis ir informacija apie jas, kuri parsisiunčiama iš duomenų bazės. Šiame darbe akcentuojama optimizacijos analizė: programavimo stiliaus parinkimas, spartaus ir patogaus žemėlapio naršymo įgyvendinimas bei duomenų laikymas pastovioje mobiliojo telefono atmintyje. Magistro darbo tikslai: surasti neužpildytą taikomųjų mobiliojo telefono programų rinkos sritį, atlikti įgyvendinamumo ir optimizacijos analizę ir analizių pagrindu įgyvendinti *Mobiliojo pirkėjo gido* taikomąją programą. Šiame darbe nagrinėjami konkurencinių programų privalumai ir trūkumai bei parenkamos *Mobiliojo pirkėjo gido* išskirtinumo galimybės. Įvairūs testai leidžia palyginti rezultatus su teorinėmis išvadomis.

Mobilusis pirkėjo gidas – tai nauja idėja bet kuriuo paros metu mobiliajame telefone rasti informaciją apie parduotuves. Išskirtinumo bei optimizacijos analizių rezultatai gali padėti ateityje kuriant panašaus tipo taikomąsias programas.

Darbą sudaro 7 dalys: Įvadas, Verslo sričių analizė, Reikalavimų rinkimas ir optimizacijos analizė, Mobiliojo telefono programavimo technologijos, Sistemos kūrimas, Išvados ir pasiūlymai, Literatūros sąrašas.

Darbo apimtis: 63 p. teksto be priedų, 10 paveikslų, 7 lentelės, 33 bibliografiniai šaltiniai. Atskirai pridedami darbo priedai.

Prasminiai žodžiai: *Mobilusis pirkėjo gidas*, įgyvendinamumo tyrimas, optimizacijos analizė, pastovioji duomenų kaupykla, daugiaprocesis programavimas, rastrinė grafika, vektorinė grafika, tinkamumo testavimas

Vilnius Gediminas Technical University
Faculty of **Fundamental Sciences**
Department of **Information Technologies**

ISBN ISSN
Copies No. …2…
Date ….-….-….

The Master thesis of Data Mining study programme

Title: **Mobile Shopper Guide: Implementation and Optimization Analysis of the Mobile Platform**

Author **Domantė Olekaitė** Academic supervisor **Algimantas Čepulkauskas, Phd.**

Thesis language

[ ] Lithuanian

[X] English

Annotation

The main goal of the Master thesis is to develop the *Mobile Shopper Guide*, a downloadable mobile phone application, which provides a city map with shops on it and relevant information once connected to the database. The focus is the optimization possibilities for programming style, fast navigation and data storage on the persistent device memory. The goals of the thesis: to find a niche for new mobile applications to evolve, perform feasibility and optimization analysis and construct the *Mobile Shopper Guide* application on their basis. "Cons and pros" analysis is performed for competitive applications and the differentiation strategy is set up. Different tests done in the end of the system development allows to compare expectations with the results. The idea of the *Mobile Shopper Guide* is a new way of presenting all relevant information about the shops at no time and always at hand. Differentiation issues raised and the optimization analysis done can be used as a guide for similar mobile applications.

The work consists of 7 parts: Introduction, Background for a Mobile Application, Requirements Gathering and Optimization Analysis, Mobile Programming Technologies, System Development, Conclusions and Recommendations, List of References.

Work coverage: 63 p. of text without appendixes, 10 figures, 7 tables, 33 bibliographical entries.

Appendixes included.

Keywords: *Mobile Shopper Guide,* feasibility study, optimization analysis, persistent data storage, multi-threading, raster graphics, vector graphics, usability testing

TABLE OF CONTENTS

TABLE OF FIGURES

ABBREVIATIONS

AMS – Application Management Software

API – Application Programming Interface

CDC - Connected Device Configuration

CDMA – Code Division Multiple Access

CLDC – Connected Limited Device Configuration

CPU – Central Processing Unit

DPI – Dots per inches

GPRS – General Packet Radio Service

MIDP – Mobile Information Device Profile

MSA – Mobile Service Architecture

NSTL – National Software Testing Labs

OC – Operating Context

OS – Operating System

PC – Personal Computer

PDA – Personal Digital Assistant

RMS – Record Management System

SMS – Short Message Service

SSL – Secure Sockets Layer

SVG – Scalable Vector Graphics

UC – Use Case

UI – User Interface

VM – Virtual Machine

VPN – Virtual Private Network

WAP – Wireless Application Protocol

WTK –Wireless Toolkit

INTRODUCTION

Relevance of the Thesis

Due to the increase of communication features through mobile phones the thesis focuses on mobile applications – installed applications rather than SMS applications or Mobile Web applications. Thesis presents the *Mobile Shopper Guide* application downloadable onto a mobile phone. The application helps the users to manage their quick "shopping routes" by providing a city map with shops displayed and relevant information about them once connected to the database. Shops can be displayed by types (restaurants, clothes, etc.). Relevant information includes: address, opening hours, special offers left by the shop owners, etc. The *advantage* of the application is that it does not require constant air-time and all needed information is accessible. What is more, people tend to carry their mobile phones with them, so they will not have to worry about not being connected to the Internet on their PCs for information. Another attractive feature is the ease of navigation, comments about the shops, like reduction alerts, special offers left by shop holders, which will allow users to be "at a right place and at a right time".

The *Mobile Shopper Guide* is a whole system, consisting of the mobile platform, administration platform and user portal. This thesis focuses on the implementation and optimization of the mobile platform.

The relevance of the thesis is justified by the growth in mobile applications business. Applications for information retrieval are preferred to wap browsers because they offer a richer user experience then browsers, which usually is painfully slow and difficult to navigate. Also because once applications are downloaded to the phone some content can reside on the device, eliminating the need for a continuous air-time or network hit with every click.

The Internet usage on mobile phones is growing at a high rate. Research by the Yankee Group (2007), the first independent technology research and consulting firm, showed that approximately the number of mobile to broadband subscribers is ten times bigger in each country all over the world, and these numbers are steadily growing. The mobile Internet is also encouraging and opening markets for new mobile Internet specific applications. „Consider how *MapQuest* and *GoogleMaps* changed the way people get directions. Now combine that with in-car GPS systems, integrate with a mobile handset, and add in search and advertising." – says Jonathan Singer, Research Analyst & Editor at Yankee Group**.** [9] The *Mobile Shopper Guide* idea is another solution to the Google Maps. The idea of the *Mobile Shopper Guide* is a new way of presenting all relevant information about the shops at no time and always at hand. Differentiation issues raised and the optimization analysis done can be used as a guide for similar mobile applications.

Novelties in Science

The *Mobile Shopper Guide* is not a completely new idea in terms of map-enabled applications; there are several similar applications already available. However, the focus on a common shopper, rather than roads, routs or address search, is a new idea. What make the application stand out is the attractive features like fast and convenient navigation, user friendly display, free tool cost, non-constant air-time, customization by the shop types. The *Mobile Shopper Guide* aims at optimal performance by means of storing only basic information the shopper needs, which results in fast navigation and information retrieval. Optimization analysis on the programming style and data storage are done, which can later be used as a guide for other similar applications striving for high performance.

Goals and Tasks of the Thesis

The thesis consists of five basic chapters, everyone of which represents separate goals. Generally, the thesis aims to develop the *Mobile Shopper Guide*, the downloadable application to a mobile phone, and motive its development decisions my means of optimization.

1. **Fields of application analysis and differentiation**. First of all, fields of application are analyzed to settle down in one of them. The task is to position the *Mobile Shopper Guide* idea among the others and determine differentiation strategy. An early feasibility study to see what is feasible is done and competitors are analyzed by features they offer in order to make the *Mobile Shopper Guide* stand out.
2. **Requirements' gathering and optimization possibilities**. Next goal is to gather the requirements and determine the tools for the application optimization. To gather the requirements and document them use cases and system architecture is developed, risks are determined, back up plans proposed and optimization analysis is done.
3. **Mobile programming technology selection**. Another goal is to compare the mobile programming technologies by means of features of interest.
4. **Application development in accordance with the requirements and testing**. Lastly, the goal to develop the application that corresponds the requirements is described in chapter five. Application development stage aims at engaging a proper programming style and developing optimal class hierarchy, conducting an early testing to further optimize the application. Acceptance and usability tests show the results to what extent the application meats the requirements and what is the end-user satisfaction.

Practical Values

The *Mobile Shopper Guide* is a new tool helping to manage the shopper information about the shops. Its differentiation strategy offers attractive features that are determined using the competitors' comparison by their pros and cons.

Several optimization analyses for the proper feature implementation are done in this work: programming style selection, persistent data storage mechanism implementation, raster image comparison and image naming convention. The early analysis and early selection of right tools and styles let the application be programmed properly. The results of the analysis can be used as a guide for other similar applications in the future.

The usability and acceptance test frameworks can also be reused for other applications to get feedback from the users.

The Correctness of the Results

The *Mobile Shopper Guide* is a fully functional application, however, it is basically a prototype. The application strives at fast performance by implementing the basic principles of the optimization. The optimization analysis results implemented allow fast performance and comfortable non constant air-time. It can be a framework for implementing future ideas and customization suggestions, because the fancy features are not likely to obstruct the performance if it is done on a right basis, compared to implementing them at once without any frameworks or analysis.

Approbation

Topic „*Specificity of Creating User-friendly System: Mobile and Web Environments*" was presented in the 11th Lithuanian Junior Scientists conference "Science – the future of Lithuania". The paper is published in the collection of the 11th Junior Scientists Conference Announcements, 2008 and is included in the list of references of the Master Thesis [1].

1. BACKGROUND FOR A MOBILE APPLICATION

1.1 CONCEPT OF A MOBILE PHONE APPLICATION

Usage of mobile phones is huge nowadays. According the research carried out by Eastern Europe Studies Centre (EESC) in 2007, 1386 mobile phones go for every 1000 inhabitants in Lithuania, meaning that almost every second Lithuanian owns two mobile phones. [10] Mobile phone has become an everyday tool not only for conversations and short messages, but is also a helpful reminder, comfortable alarm clock, notebook, radio, web browser… thing you really need throughout the day, that is why a mobile phone is a thing that is always with you.

To make people's life even better and to fulfil even more needs, a variety of mobile applications has emerged. One can gamble, buy tickets, perform financial transactions or join chats systems via a cell phone. There are five main mobile application types:

- Communications
- Information access
- Mobile computing
- Business solutions and mobile commerce

Each of them has different population and targets different user groups. Communications and information access applications are used by all user groups on a daily basis. These applications are for everybody for keeping in touch with the world around us. Computing and Business process solutions, though, are more specialized areas. Mobile computing applications are usually developed for a specific project and implement some complex algorithms, like shortest path calculations or medical diagnosis based on patient's symptoms. Mobile applications add value to Business processes by increasing communications and knowledge sharing. Moreover, mobile commerce (m-commerce) expands the ways business is done and brings extra profit.

Usage of mobile applications has been growing each year. According MobileMoves, the mobile news web page, growth in revenue for mobile business applications is expected to increase by 44 percent in 2008. In year 2007 it was close to 50 percent increase. In-Stat Analyst Bill Hughes predicts the four general applications, wireless email, wireless Internet access, wireless instant messaging, and personal information management, will have the highest penetration since they are easier to create. [2]

UK Mobile phone web log gives the list of the most popular mobile phone applications in year 2007. Naturally, they belong to the first two mobile applications categories, communications and information access:

- **Gmail** mailbox via your mobile wap browser, including all standard features, like attachments, archives, etc.;
- **Opera mini** – slimmed down version of Opera browser;
- **Yahoo Go**, providing access to various services, like Yahoo Calendar, OneSearch, Mail, Sports and News, etc.;
- **Yell**, allowing to search for a specialist, like dentist, chemist, florist, or search for a hotel, gift or just an advice, by simply connecting to the Internet and getting the information needed;
- **Shozu**, which is an application allowing user to upload pictures to the web log for various online services available, like Flickr, Blogger, Picassa, etc.

Apparently, general information access applications (like wap browsers) are the most widely used. The customization and improvement of these kinds of applications has gone really far.

Next thing what a common user, still hungry for information, would think of is a more specified information retrieval. It is quite difficult to search for a dentist over the Internet unless you know the exact URL. The fourth application in the most popular applications list covers this issue by providing a convenient and quick search engine helping to find a specialist. When the information thirst is satisfied, people are tend to communicate with each other by sharing photos, which is the fifth bullet in the list of the most popular applications.

The need for specific information is an important, but the least developed area. The browsers can hardly be customized more. Search engines for different information, on the contrary, can be customized endlessly, since users are so different and thus their information needs.

1.2 FIELDS OF APPLICATION

Mobile phone applications are used in variety of business areas as well as of everyday life.

It can be used in medicine. In many fields of medical treatment and healthcare, a monitoring of certain body functions is essential. For patients suffering from diabetes, a regular check of the blood glucose level forms is required at a daily basis. That is why in many cases portable medical devices are used. In 2006 a certain medical software download to a mobile phone was introduced. It was meant for medical data monitoring via a handheld device, like monitoring glucose level information, a blood pressure information, a cholesterol level information, or a coagulation information. [11] In everyday life you may need an application in your mobile phone with a map and possibly a GPS receiver to automatically display the map of your current position (e.g. Mobile Gmaps). [12]

Mobile entertainment, interactive games, betting, electronic books, dictionaries are also used within a single mobile device. Mobile shopping, reservation and ticketing, mobile payment, billing and

advertising are also widely spread. For example, Citibank in California introduced *Citi Mobile*, the first downloadable mobile banking application from a major financial services provider. After connecting to the Internet and downloading the application to a mobile phone, customers can view balances, pay bills, transfer money, locate ATMs, and click to call customer service. Since the *Citi Mobile* application resides on the phone, it is faster and offers a graphics intensive interface, customers can select the bank icon on their phones to access accounts instead of navigating through multiple Web pages on a tiny screen. [13]

1.3 PROGRAMMING TECHNOLOGIES IN MOBILE PHONES

A variety of mobile phone programming languages and tools has emerged, differing in popularity depending on the application needs, platform requirements and facilities that the tools offer.

Table 1 provides a brief overview of currently the most popular mobile programming technologies. [14]

Table 1. Most popular mobile programming technologies

Programming tool	Comments
J2ME	The most popular and ideal for a portable solution. It is suitable for many types of mobile phones, because device-specific libraries exist for many devices.
Symbian	Symbian is very powerful for general purpose development. It is strongly supported by Nokia. Other types of mobile devices have limited support (Fujitsu, Sony Ericsson, Mitsubishi, Sharp and partly Motorola). Spread in Europe and Japan with little penetration into US market.
Android	Android is a quite new Linux-based platform. It is supported by 34 major software, hardware and telecoms companies, therefore it is likely to be rapidly adopted from 2008. Programming is done in Java.
Lazarus	Meant and is ideal for prototyping and quickly developing database powered applications. Lazarus is also useful for porting Object Pascal software to mobiles.
Python	Phyton is ideal for initial prototyping and concept testing when functionality falls outside Java ME.
Brew	Brew is ideal for deploying applications for deployment on CDMA-based networks (also supports GPRS/GSM models). Little penetration in Europe.

Software programming tools for mobile phones support different application types and different mobile phones. It is up to the programmer which one of them to choose. Java2 Micro Edition (J2ME) might be preferable if targeting many different mobile phones models. Non-Java platforms (e.g., Symbian) are created for a smaller range of devices, thus a platform vendor can have tighter control over implementation diversity for every platform. However, developers may still have to develop a J2ME equivalent as well, if a wider range of platforms is to be targeted. Because J2ME runs on a Java virtual machine, it requires more processing power than, e.g., Brew. Whereas J2ME

is language independent, Brew applications can be developed in C, C + + or Java, depending on the host device, thus Brew-enabled devices with a Java virtual machine are capable of running Java applets.... All of these and other significant aspects have to be considered prior to choosing the optimal means for application development.

1.4 RESEARCH OUTLINE OF THIS WORK

With the idea of a mobile application concept, application field variety and technology overview, further in this work a more deep research in mobile applications implementation fields and their relevance is done. As concluded in the first chapter, the least developed field for mobile applications is specialized-information access.

Next, the application idea of a *Mobile Shopper Guide* will be promoted, and the choice motivated. Further, a feasibility study will be done to find out what is feasible and what is not. A research will review already existing similar and possible ways to differentiate.

Requirements' gathering is a next stage, involving use cases analysis, requirements of functionality, hardware and software, system architecture analysis, early GUI modelling, risks overview and initial optimization analysis.

Further in this work mobile programming technologies is analyzed by means of their features, like availability, platform suitability, graphical interface, etc., and the most proper is chosen. Afterwards the system development process is described with special focus on optimization analysis.

To conclude, further outline is business field selection, feasibility study, requirements' gathering and tools selection to best support the idea and the development process itself.

2. BUSINESS ANALYSIS

2.1 MOBILE PHONE APPLICATIONS CATEGORIES AND THIER RELEVANCE

As mentioned in chapter *Business Analysis*, Communications and Information access areas are the most popular since they occur in everyday life. *Communication* applications on mobile phones mostly target youth who enjoy using various chat systems for chatting, making friends, etc. For instance, *Symmy Instant Messenger* allows talking, sharing files and sending and receiving messages via application on a mobile phone. *SMS Chat System* offered by *Tonic Mobile* presents the ability to chat with several people simultaneously, send multiple videos and pictures. [19] A lot of freeware applications are available also, like *okChat* offering a "chat room" with numerous accounts with their own login features. Communications area is limited in terms of functionality and completely new features are hardly possible with current variety offered already.

Information access is a wider area. People like getting refined information. While on holidays, travelling related information is of interest. Such information providing applications may specialize on works of art look-up (paintings, statues, buildings), historical info or tour guide. Others may offer currency look-up, building address look-up or sign translator abilities. After work people are interested in shopping information, like what shops are still open and how to get there as quickly as possible. For leisure activities people need movie look-up abilities, restaurant guide or real estate look-up. This area targets all user groups.

Mobile computing in general describes ability to use technology "untethered", i.e. not bound to handheld devices having access to a shared data stores independent of the physical location. Mobile computing applications are based on computational algorithms and are used in experts' knowledge requiring fields – medicine, optimal travel routes, etc. A natural language voice interface for messaging, address book or calendaring is among such mobile applications. It involves speech recognition though natural language understanding and text-to-speech conversion. Other kind of computing applications is context-aware applications. Location-aware applications are those in which the location of a specific person or an object is used to shape the application. For example, driving assistant that alerts a driver to traffic congestion ahead and suggests an alternate route. [20]

Personal Care Connect application introduced by IBM provides health care sensors attached to a mobile phone platform via Bluetooth. Sensor readings are collected from the health care sensors and are converted into a normalized event format on a mobile phone. These events are injected into an event engine running on the phone, which redistributes the events to subscribed applications running on the phone. [21]

Mobile applications employed for *Business solutions* can benefit a lot. A mobile workforce can increase productivity and bring financial success to a company. For instance, Nokia introduced several mobile business solutions. Nokia's *One Business Server* is a solution for mobile email, which leaves the company freedom to integrate any email environment to the mobile device. It forces the corporate network content to fit on small screen, making no need to change existing applications. Nokia's *Secure Access* system is SSL-based VPN solution, which enables secure access to corporate email, applications and resources for mobile employees and partners. [22]

Mobile payment applications, belonging to *Mobile marketing*, or *Mobile commerce*, allow personal payment transactions like ticket purchase or local merchant payments. Mobile ticketing for airports or train stations helps to streamline unexpected metropolitan traffic. It also helps users remotely secure parking spots. Location based mobile services use the location of the mobile phone user as an important piece of information used during mobile commerce transactions. These services are, e.g. local maps, local offers, weather, etc.

As mentioned in the previous chapter, most popular applications fall in the category of applications for everyday usage, especially for refined-information retrieval. This work will focus on applications falling to this area, because despite plenty of solutions offered, yet lots of unimplemented ideas are still available.

2.2 A MOBILE SHOPPER GUIDE

A lot of people visiting new cities are first of all interested in visiting famous and noble places. Next thing they look for is shops. Shops to buy clothes, food, souvenirs and books. Even native inhabitants like to go shopping after work or during their weekends. It is hard to decide in advance of which shops to visit, therefore, looking for information via the Internet over PC makes no sense. People fall for spontaneous decisions. But how to know which shops offer best prices and discounts, or which restaurants are worth visiting that day when deciding spontaneously? Imagine a mobile application offering an interface with a city map and shops on it with their related information, like opening hours and discount alerts. Imagine reaching all needed information via a simple mobile phone, device that people carry with themselves almost all the time. The idea of a *Mobile Shopper Guide* is a mobile phone application, displaying a city map with information about the shops retrieved form the database. The user can navigate the map, select certain shops and get the information about them within no time – address, opening hours, telephone number, special offers for the day or discounts.

The *Mobile Shopper Guide* aims at providing cheap and comfortable way to manage the "shopping" information. Most of the nowadays map-related applications provide detailed information, but only when connected to the web and therefore it costs to use it every minute. The idea that this work

focuses on is the minimal usage of the Internet connection for user convenience and satisfaction. In the idea is that the user still has to receive updates via the Internet, but only once – to get the list of all shops in the database and constant air-time will not be required afterwards. The user will always have the map with him/her and will do the updates only to check if any new information is available at his/her convenience.

The idea of having a map on a mobile phone is not completely new. There are several similar applications already available. They offer different advantages and lack different features. See competitors listed in *Appendix A: The Competitors*. List displays their main pros and cons, which were analyzed and reflected in this work. The Appendix lists both, substitutes and alternatives. The list also considers an ordinary map as a substitute, which can be a paper city pocket guide. Several downloadable electronic maps are also analyzed, some of which are alternatives rather than substitutes, since they are maps accessible via the wap browser and not really the applications. Having looked into the pros that the competitors offer, in this work we aim at providing advantages over the competitors, by offering these differentiations and added values for the user:

- Fast
- Easy to update
- User friendly
- Free for the user
- Constant air-time not required
- Can be customized according shop types
- Easy navigation

2.2.1 Values for the User and involved Companies

Supporting the *Mobile Shopper Guide* idea, ways to attract the user have to be considered. First of all, we strive at providing *fast navigation*. Today more than ever, with growing system resources and technologies that offer easy and fast access to any shared information users are impatient and eager to get information within few winks of an eye. Therefore, we aim at optimizing the performance of our application, which is tightly related to storing only basic information onto the phone memory which is not overloaded with unnecessary (as we see it) information.

Having all basic shop information in your mobile phone is very convenient and flexible. You don't have to worry about connecting to the Internet on your PC at home to check the opening hours or special offers which may be time consuming. Visitors from foreign cities don't need to bother about buying pocket guides; they can simply download the application from the Internet and have it with themselves all the time. The easiness to *use* and *update* is another advantage.

What is more, updates and shop display does not require *constant air-time*, since all information about the shops is kept in persistent memory of a mobile phone.

The "shopaholics" are able to read necessary information about shops, like opening hours, contact information (telephone, person in charge), any special offers or notes left by the shop holders, such as discounts, "one-for-twos", sales, etc. The user will not simply get the address of the shop, he/she will be provided with *directions* indicated by arrows as of which turn to go. Moreover, a shopper will be provided an ability to specify what *kinds* of shops to display on the map. If the user is only interested in finding shoe stores, a functionality to list only shops that sell shoes will be provided. A feature to zoom in and out of the map, in order to see the streets better, is provided also.

The application targets mostly *mobile phones*, not PDAs. From this perspective, mobile phones are much cheaper than PDAs and most of the people already have their mobile phones [9]. A PDA is more expensive and the user would probably be afraid of loosing it and carrying it all the time. The service we propose will not require any specific features for the mobile phones; the minimum requirements should be a Java-compatible phone with a 128x160 colour-screen, GPRS and WAP connection.

Striving for a cheap solution, the application could also be *free* for the user. Business idea is to charge the shop owners who want to advertise in the *Mobile Shopper Guide* application by requesting the coordinates for the shop on the map and leaving comments as for special offers and discounts.

2.2.2 Technology Supporting the Idea

First thing before starting the application development is a quick look at the technology available that would support the idea. The *Mobile Shopper Guide* is a whole system, not a single isolated application within the boundaries of a cell phone. A database to retrieve the updated from, administrative tools to add/ delete shop locations and comments must be integrated together. For this purpose an early *feasibility study* must be carried out.

The main goal of the feasibility study is to identify main technical requirements and for the mobile and supporting platforms and to decide is it feasible from implementation point of view and how wide the applications can be supported. It includes requirement gathering and investigation on what is achievable and what is not.

The first step is connecting to the web page via mobile phone or PC to the web page where the jar file is stored. This can be is done via the *GPRS* and *WAP*, or via the Internet connection and Bluetooth afterwards. Once the user downloads the jar file (the shopper guide application) to their java compatible cell phone (if java is chosen as a programming tool), it is kept in phone memory. When user launches the map, he/she can then download the shop list. This is done by establishing

the connection with the *Database Server*. The java application connects, possibly, with the *SQL Server* and checks for updates available. If new information is available (which is always true for the first time connection), updates are downloaded into the mobile phone. User can navigate the map and zoom in and out. The user can select only certain types of shops to display. Shops are represented by icons (raster images). The user can navigate between them and cursor-over them to get information on what shop it is. For newly emerged shops location information can be added to the database, as X and Y coordinates. To do this the system administrator must connect to the administrative platform and commit changes to the database. Once changes are committed, users can update the new information. User should be able to read extensive Help instructions both, locally and on the Internet. Therefore, a *User Portal* is required to store the information in *htmls* and *wmls*.

Talking about the mobile phones targeted, basically they are the ones, released in 2004 or later. This is because the main purpose it to attract as much users as possible, and mobile phones models released in pre-2004 are not as widely used.

2.3 CONCLUSIONS

This chapter overviewed a great variety of mobile phone applications and the most popular and desirable were distinguished. Information retrieval it is a good medium for new ideas in a form of mobile applications to emerge.

The idea of *Mobile Shopper Guide* was raised and positioned among other similar applications by means of differentiation strategy. An early feasibility study showed that the idea does not require any special hardware requirements and software tools. However, the idea is not brand new, therefore, already existing solutions of a similar kind were analyzed in terms of the advantages and disadvantages and new added values that could attract the user and make the application different were posed. These include fast navigation, performance optimization, easiness to update and the advantage of a non-constant air-time.

Further the work will concentrate on the requirements' gathering for the *Mobile Shopper Guide* system in order to implement the idea on time, satisfy the functionality and fulfil the goals posed.

3. REQUIREMENTS GATHERING AND OPTIMIZATION ANALYSIS

3.1 FRAGMENTATION OF MOBILE PHONES

For the application development, it is necessary to define the target platform first. An important issue has to be considered – the *fragmentation*. Fragmentation is the inability to "write once and run anywhere", in other words, to develop an application that would fit and have the intended behaviour on all platforms, or operating contexts (OC). [23] Damith C. Rajapakse (2008) in his article investigates the fragmentation of the installed mobile applications and means of finding the reasons behind it and ways to tackle it.

Fragmentation is caused by the natural diversity of the platforms, or OCs. The diversity itself can be classified as:

- **Hardware diversity**. This kind of diversity is notable for the mobile guide, displaying maps, since such differences as screen size, colour depth or orientation is an important issue. Memory size, processing power or connectivity are key factors for map-displaying applications to work correctly.
- **Software diversity**. Platform differences, such as Symbian, Nokia OS, PalmOS, Mobile Linux, etc., API standards (MIDP 1.0, MIDP 2.0, etc.), proprietary APIs, differences in multimedia support (such as codecs), maximum binary size allowed influence the application behaviour.
- **Feature variations**, such as *beta* or *light* versions versus full versions may also impact the behaviour.
- **User preference diversity** includes language, style or accessibility requirements.

Obviously, most of the diversifying factors depend on a particular mobile phone model. It is important to know about the fragmentation, because it influences all of the application development cycles, like business modelling, requirements' gathering, analysis, design, development and testing. First of all, in the business modelling the optimum set of OCs has to be figured out, one has to select the suitable platforms, and neglect ones that require too much effort for fitting the application with the respect to their representing market size. Requirements' gathering stage will naturally focus on the selected platforms. An in-depth knowledge of the OCs might be required, since it may turn out that different OCs results in exceptional and alternate flows. When in the design stage of the system architecture, the design needs to be able to accommodate not only the targeted OCs, but also the future platforms. However, targeted OCs are the most important. What concerns testing, emulators are not enough for functionality verification. All targeted platforms must be tested to verify they work as intended.

All of the factors put together, it is impossible to have an application that will fit all platforms. Practitioners say that sometimes a single application can end up having 400 or even more different versions.

As for the *Mobile Shopper Guide,* the biggest concern of the diversification is the screen size and the persistent memory on different devices. One of the solutions to reduce the risk is the elimination of the *accidental* diversification, which is diversity due to API implementation bugs, unintentional, avoidable and hard to predict. Means for the elimination are more detailed specifications, using technology compatibility kits, platform independent programming technologies. Major companies in the mobile application industry have a significant role here. Such an effort in the Java ME arena is the *Mobile Service Architecture* (MSA). MSA is a standardization of APIs aiming at development and deployment of the widest possible variety of applications, in a form that will be easily portable across a big spectrum of mobile devices. [24]

Can Java technology be a solution to minimize fragmentation? Theoretically, a Java ME application is able to run on any Java-enabled mobile device, meaning that it can target much more OCs than a non-Java application. Non-Java platforms (e.g., Symbian or Brew) are created for a smaller range of platforms, in this way a tighter control over implementation diversities is available, which reduces the diversities within a single platform. However, a Java ME equivalent might be needed as well if a wider range of OCs is to be targeted. Fragmentation in, e.g., Android platform appears to be less at the moment, but this could be simply because it is not yet used in a wide range of OCs.

3.2 FEASIBILITY STUDY

Aware of the fragmentation limitations, we have to figure out how feasible the implementation of the *Mobile Shopper Guide* is and what platforms shall it target. This is done by the *feasibility study*. As briefly overviewed in chapter *Business Analysis*, feasibility study means determining project's viability and the conclusions of the study tell either to proceed with the project or leave it is made. Feasibility study must be done for the deliberation of the project team before the requirements' gathering. The conclusions of the feasibility study suggest recommendations, risks and limitations and prepare for the whole system development having in mind implementation assumptions and possible risks. This study is the first time in the project when pieces are put together to see if they can perform together in a reasonable way. The study shows the sensitivity and impact of the assumptions. [25] In the feasibility study competitive advantages and competitive applications must be reviewed. The competitors list by their features and "the cons and pros" was briefly reviewed in chapter *Business Analysis*. Also, feasibility study includes the exact project definition, of what is going to be implemented, and what mobile phones to target. This includes use cases definition and mobile phone suitability determination.

3.2.1 Use Cases

One of goals of the feasibility study and requirements' gathering stage is the uses cases (UC) overview.

UC1 "The mobile updater"

1. The user accesses the web portal on his/ her mobile phone using Bluetooth - and Internet enabled PC.
2. The user can then choose to download the application or to update the application.
3. If the user moves out of range from the transmitter it must be possible to revert to the old information that is not updated.

UC2 "The interactive updater"

1. The user starts the mobile application while in the city, not having access to a PC.
2. The application asks the user if the client database should be updated.
3. The client connects to the database server using GPRS.
4. The client has possibility to discard the updates.
5. If the connection is broken it must be possible for the client to perform a rollback reverting to the old database.

UC3 "Money maker"

1. The user feels the need to withdraw some cash. The user might be new in the city and have no idea where to find an ATM.
2. The user chooses to view only banks and ATM machines in the client application.
3. On the map banks and ATM machines are shown as clickable icons.
4. When a user clicks on an icon information about the bank is shown, such as the name of the bank, opening hours and if foreign currency is available.

UC4 "Feed the hungry"

1. The user gets hungry and wants to know where the nearest restaurant is located.
2. The user chooses to open the application and view the restaurants located in the area.
3. The application points out different restaurants on the map, as clickable icons.
4. When the user clicks on the icon he/she is interested in the information of that particular restaurant is shown.

UC5 The "shopaholic"

1. Desperate for some new clothes, a "shopaholic" is interested in what shoe shops in Vilnius can be found within a reasonable distance.
2. The "shopaholic" downloads the application from the webpage and uses it.
3. The "shopaholic" chooses the "shoes" type and gets a map showing all the shops in the area represented as little icons.
4. The "shopaholic" then chooses an icon from the map and gets all the important information the shop has entered, including sales and offers.

It is obvious that a Shopper Guide is not an isolated mobile platform. A database server to store information about shops, administrative platform to enter and monitor the information and a user portal for the application download and a help is required.

3.2.3 System architecture

Use cases' determination helps to identify main system parts and depict a rough system architecture diagram for the *Mobile Shopper Guide* (see Figure 1).

Three main system parts are: the *mobile platform*, *administrative platform* and the *user portal*.

Mobile platform. At first, the user connects to the web page where the application is stored. This is done via the GPRS and WAP. The user downloads the map application to their java compatible mobile phone. The java application is stored in the free memory of the cell phone. When the user starts the application he/she can update the shop list, getting the information from the database. When the user first downloads the application there will be no information about the shops, this has to be downloaded from the database. After the shop list is downloaded, the user can navigate the map and zoom in and out for a better view of the map and shop locations. The user can choose to display only certain type of shops. The user can navigate between the icons which represent shops and selected one to see the relevant information about the object.

Administrative platform. The client (a shop holder) sends in information to the database by connecting to the web server. The administrative platform provides a web page for the clients to enter information about the shops. Depending on the login type (administrator or non-administrator), the administrative web site grants different rights – administrative or user. There is a map where the shop holder can pin point the location of the store on a similar map that's used in java application. The information is maintained by the system administrators, who will have to accept the clients and maintain the information that the clients put in to the database.

User portal. User portal is a web page from where the user downloads the application form. This page is separated from the administrative platform and the updates. On this web page the user can read about the application functionality and decide if it's suitable for their mobile phone model. On

the server there are two separated files, a JAR and a JAD file. For some phones both files are required (not only a JAR file), otherwise the phone will not work correctly with the application. The files can be downloaded to your computer or to the mobile phone directly. In the first case, the user transfers them to the mobile phone with Bluetooth, IrDA, or serial cable. The user downloads the application by using GPRS and to make this simpler for the user, the user portal will also have a WAP page where the links to the files are shown.

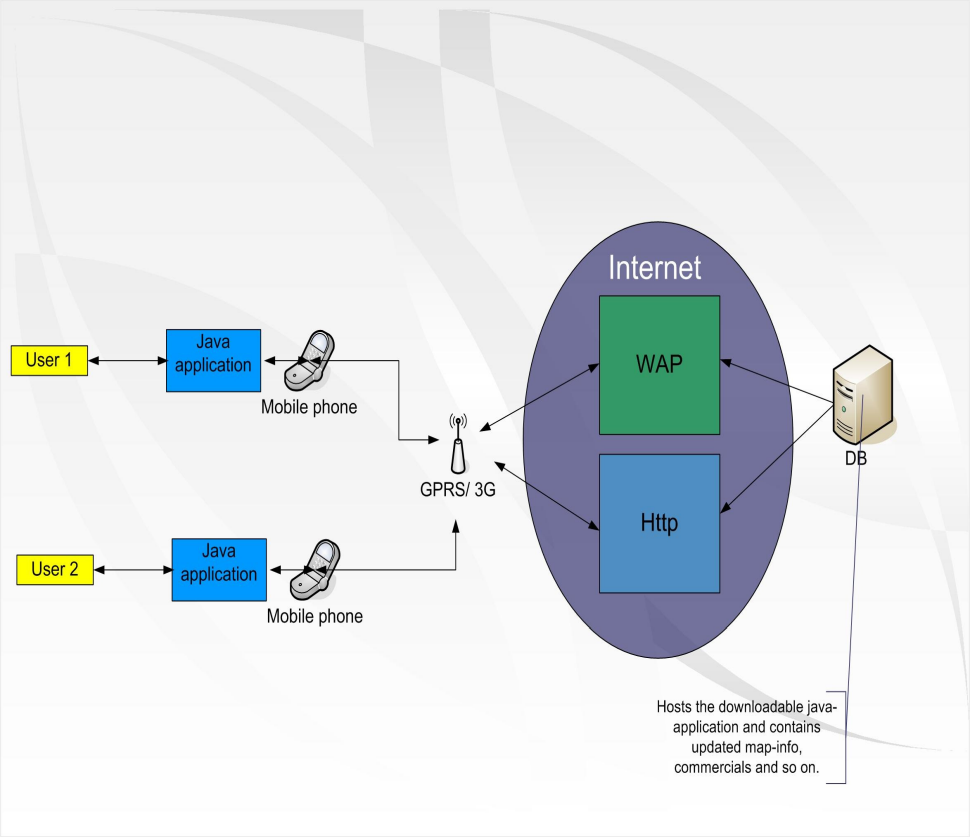


Figure 1. The initial *Mobile Shopper Guide* system architecture

3.2.4 Risk Assessment

Risks always exist in projects and it is necessary to identify them so that they can be considered and mitigated in time. There has to be a back up plan to minimize the risks so that they don't damage the project. For this we need to first identify the potential risks and examine how big the risks are and make a plan on how to minimize them. In this work Sommerville's example was taken for the risk management and the risk analyzing structure (Sommerville, 2001). [3] First of all the most catastrophic risks are identified, which are as follows:

- The j2me application is too big for the mobile phones we are using.

Solution: Restrict the number of POI (points of interest) your application could have and bring the quality of the map down.

- The j2me application can't communicate with the database correctly.

  Solution: The updates will not work; the shops and the information about them will not be displayed on the map.

Table 2 includes a more detailed risk assessment, grouped by risk types.

Table 2. Risk identification by risk types

Risk Type	**Possible risks**
Technology	• The j2me application is too big for the mobile phones we are using. • The j2me application can't communicate with the database correctly. • The maps are not visible enough on our mobile phones.
People	• Inexperienced people, people who don't have enough skills to accomplish the tasks planned.
Tools	• Netbeans is not efficient enough for our kind of application.
Requirements	• The requirements are not evolved enough. • The requirements may change during the project development.
Estimation	• The time schedule and the milestones are not achievable. • The size of the software is underestimated.

Next step is to evaluate the likelihood of the risks. Table 3 evaluates the risks by probability (column *Probability*) and severity (column *Effect*). Probability evaluation criteria Low to High, Effect is estimated using Tolerable-Serious-Catastrophe scale.

Table 3. Risk likelihood analysis

Risk	**Probability**	**Effect**
The j2me application is too big for the mobile phones we are using.	Medium	Catastrophe
The j2me application can't communicate with the database correctly.	Medium	Catastrophe
The maps are not visible enough on our mobile phones.	Medium	Serious
Inexperienced people, people who don't have enough skills to accomplish the tasks that we need.	Medium	Catastrophe
Netbeans is not efficient enough for our kind of application.	Low	Serious
The requirements are not evolved enough.	High	Serious

The requirements changes during the project.	High	Tolerable
The time schedule and the milestones are not achievable.	Medium	Serious
The size of the software is underestimated.	Medium	Serious

Every risk must have a back up plan in case it really occurs and stops the project from further development. Table 4 lists all risks again and decides the strategies of what to do to mitigate the risk.

Table 4. Risk mitigating strategy

Risk	**Type of strategy**	**Strategy**
The j2me application is too big for the mobile phones we are using.	Minimization strategy	Restrict the number of POIs in your application and bring the quality of the map down.
The j2me application can't communicate with the database correctly.	Contingency plan	The updates will not work; the shops and information about them have to be within the application.
The map is not visible enough on our mobile phones.	Minimization strategy	Delete the unimportant stuff and try to highlight the roads on the map.
Inexperienced people, people who don't have enough skills to accomplish the tasks.	Avoidance strategy	Get help from the teachers and the tutor.
Netbeans is not efficient enough for our kind of application.	Minimization strategy	Other environments can be used, like Eclipse.
The requirements are not evolved enough.	Avoidance strategy	Ask ourselves if we have requirements that are enough and satisfying for our project.
The requirements changes during the project.	Contingency plan	Adapt to the new requirements and save the old work; it could still be useful for the project.
The time schedule and the milestones are not achievable.	Minimization strategy	Cut out functions that are not necessary, if we don't have time to implement this function in the application.
The size of the software is underestimated.	Minimization strategy	Same back up plan as the suggestion above.

3.3 REQUIREMENTS

Requirements' gathering follow the use cases and initial system architecture notes set up in the feasibility study. Kruchten (1999) classifies all requirements into two main parts: *functional* and *non-functional* requirements. The functional requirements refer to the actions the system must be capable to perform. Non-functional requirements are related to the end-user. [4] *Technical* requirements mean hardware specification that is necessary to certain goal to achieve. *User requirements* are minimal hardware and software requirements for the user, the owner of the application.

The application itself does not have any specific hardware requirements for the mobile platform (except the screen resolution), so basically many kinds of mobile phones will be targeted. From this point of view J2ME as a programming technology is most suitable since java enabled phones are most widespread. The list below depicts the technical, user, functional and non functional requirements gathered for the mobile platform.

Technical requirements:

1.1 Java-enabled (J2ME with MIDP 2.0 and CLDC 1.0)

1.2 Colour-screen, standard resolution should be 128x160 pixels, but it will be expandable.

1.3 At least 3 Mb of internal memory and the application should be within 1mb.

1.4 An apache server that runs the servlet that connects with the database.

1.5 GPRS

1.6 Wap 2.0

User requirements:

1.1 A java enabled phone with a colour screen and enough free memory. It should also have GPRS, WAP and be compatible with MIDP 2.0 and CLDC 1.0.

1.2 The user usually needs administration rights to install the java application on the mobile phone.

Functional

1.1 Start the application

1.2 Establish a connection with the database

1.3 The user accepts the confirmation for the updates

1.4 Updates the application

1.5 Change the types of shops to display

1.5.1 Scroll between the types of shops

1.5.2 Accept a single type of shop

1.6 Move a pointer to navigate the map

1.6.1 Move the pointer around a map overview

1.6.2 Move the pointer around a detailed map

1.7 Click with the pointer on a point of interest

1.7.1 Show shop name

1.7.2. Show address

1.7.3 Opening hours

1.7.4 Telephone number

1.7.5 Comments

1.8 Show help

1.9 Quit the application

2.1 Scroll between all points of interest

2.1.1 Show where the POI resides on the map

2.2 Keywords to find the right shop types

Non-functional

1.1 The user should get the latest updates

1.2 The same commands and functions should be accessible on different mobile phones models

1.3 Many users can update at the same time

1.4 The update should not take more than 10 seconds

1.5 It shouldn't take more than 5 seconds to scroll on the detailed map

3.4 EARLY USER INTERFACE DEVELOPMENT

After functionality is settled it makes sense to set up an initial Graphical User Interface (GUI) of the application, the "baby faces" of the applications screens.

The initial GUI was designed having in mind the functionality set up, the feedback that must be provided to users, like message on successful updates and feedback that is needed from the users, like confirmation to update. One of our guidelines when creating the interface was to comply with Shneiderman's and Plaisant's eight golden rules (Shneiderman & Plaisant, 2005). The eight golden rules are principles to use when making user interfaces. These principles are quite simple, but they are a great platform to stand on when you start designing a user interface. Even if our use of the principles were used as a guideline, it was still something that made our decisions easier to make. These eight golden principles can be summarized as:

- Strive for consistency.
- Enable frequent users to use shortcuts
- Offer informative feedback.

- Design dialog to yield closure.
- Offer simple error handling.
- Permit easy reversal of actions.
- Support internal locus of control.
- Reduce short-term memory load. [5]

The initial prototype of the application GUI see *Appendix B: The Early GUI of the Application*. Each picture corresponds to a mobile screen and is described by what user actions are taken.

3.5 OPTIMIZATION ANALYSIS

Aiming at developing an application that is optimal, fast, with no time lags, easy to navigate and use, before the system development is even started initial optimization analysis has to be done. This will help to address the significant issues like class hierarchy structure, the best design pattern for class development, class states monitoring, map images and mechanism to store the data on the mobile phone memory. The following paragraphs address those issues in detail.

3.5.1 System Design and Class Structure

Firstly, for an optimal system performance *thread-based* architecture is considered. Software speed optimization highly depends on reducing the execution duration for some piece of code by designing and implementing the software in an appropriate manner. Therefore, there are several reasons for choosing thread-based architecture. Modern languages and operating systems often encourage developers to use threads to mask the overhead of some operations and simplify program structure. Threads can reduce the overall execution time of threaded programs by 15-30%. For the *Mobile Shopper Guide* application thread-based architecture was chosen. First of all, there must be a thread related to the database connection, always waiting for a request to establish the connection. Another thread is related to the display of the map. Since the application will be zoomed in and out constantly, a thread to manage the changes of the display has to be implemented for a steady workflow.

Another consideration is the management of the *key-repeat speed*, which in some phones can be very slow. Therefore some piece of code should consider optimization of the key-repeat speed when navigating and respond to the user input at an appropriate rate.

What is more, class *instances* should be managed properly, created and destroyed when not used anymore to free the memory. For this purpose, a proper design pattern must be engaged. *Singleton* pattern was chosen as the best solution. Singleton pattern is used to restrict instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions in the system. [15] The singleton pattern is implemented by creating a class with a method that creates a new

instance of the class only if one does not exist. If it does exist, the method simply returns a reference to that object.

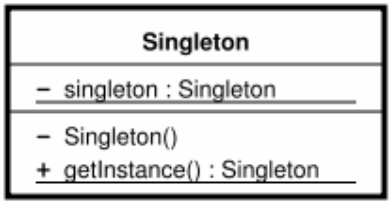


Figure 2. Singleton class

In this case a single map display class object has to be kept. When navigating the map, images change fast and so do the position coordinates. Current view has to be managed sensibly. Creating a new object for each map display makes no sense. More reasonable is to keep the same object and to pass X and Y coordinates of current position and the displayed map as parameters. This would save a lot of memory and time.

Why Singleton? Using a global object ensures that the instance is easily accessible but it doesn't prevent from instantiating multiple objects, since you can still create many local instances of the same class. The Singleton pattern provides a solution to this problem by making the class itself responsible for managing its sole instance. The sole instance is an ordinary object of its class, but that class is written so that only one instance can ever be created. In this way it guarantees that no other instances are created. [26]

However, Singleton pattern has to be used very carefully in case multiple threads exist in the class. It may happen that two threads are trying to access the same piece of code at the same time, which may result in two objects created. It objects the Singleton idea of maintaining only one object throughout the program lifecycle. Therefore, special means have to be considered to prevent that from happening, if Singleton pattern is still to be used. Based on our assumptions to have one thread dedicated to database connection and one or two for creating and managing the map displays, there shouldn’t be problems of two objects appearing unexpectedly. In other words, our assumptions let us suppose that the application architecture will be *single-threaded*.

Next, we have to consider how to optimize Singleton for single-threaded applications. One issue to consider is the destruction of the created objects. The instance must be manually destroyed by calling delete before the application terminates. Otherwise, a memory leak will be caused. Also undefined behaviour is brought in, because Singleton's destructor never gets called. According Danny Kalev (2004) single-threaded applications can avoid this by using a local static instance instead of a dynamically allocated one. [26] This is because unlike dynamically allocated objects, static objects are destroyed automatically when the application terminates, so the instance doesn’t have to be destroyed manually.

3.5.2 Map as an Image

Another important issue for optimizing the application is the map image. Firstly, the map can be represented in two ways: as a simple *raster* image, or a *vector* image.

Rather than being composed of pixels, *vector graphics* consist of points, lines, and curves which, when combined, can form complex objects (Figure 3). These objects can be filled with solid colours, gradients, and even patterns. [27]

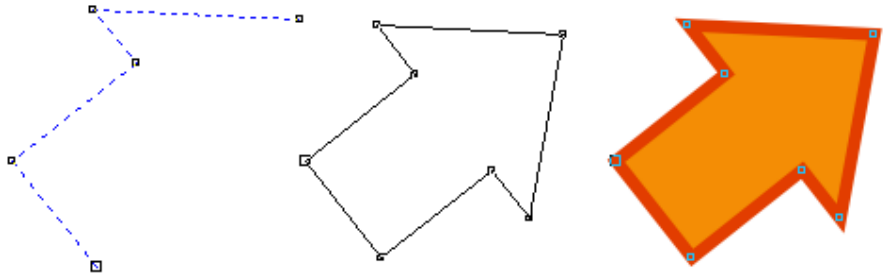


Figure 3. Vector graphics example

Advantages to this style of drawing over *raster graphics*:

- This minimal amount of information, hence, much smaller file size compared to large raster images, though a vector graphic with a small file size is often said to lack detail compared with a real world photo.
- On zooming in, lines and curves need not get wider proportionally. Often the width is either not increased or less than proportional. On the other hand, irregular curves represented by simple geometric shapes may be made proportionally wider when zooming in, to keep them looking smooth and not like these geometric shapes.
- The parameters of objects are stored and can be later modified. This means that moving, scaling, rotating, filling etc. doesn't degrade the quality of a drawing. The map, however, is not very likely to be changed in the future, therefore this advantage is not relative for the *Mobile Shopper Guide*.
- From a 3D perspective, rendering shadows is also much more realistic with vector graphics, as shadows can be abstracted into the rays of light which form them. But the 3D graphics is also not relevant for the shopper guide application, the map is only 2D.

Next step is to decide which format of raster image is the best to use, that is which takes less space while still retaining the quality of the map. Possible formats are GIF, JPG, PNG, BMP and TIFF. Generally, image file sizes, expressed in bytes, increase with the number of pixels in the image, and the colour depth of the pixels. The more pixels the greater image resolution and the file size. Also,

each pixel making up the image increases in size as colour depth is increased. An 8-bit pixel (1 byte) can store 256 colours and a 24-bit pixel (3 bytes) can store 16 million colours. [16] Table 5 reviews the differences in file formats and what they are best suitable for.

Table 5. Raster file formats

Format	Colour depth	Other features	Suitability
JPG (Joint Photographic Experts Group)	Supports 8 bits per colour – red, green, and blue, for 24-bit total – and produces relatively small file sizes	Lossy format - JPEG files can suffer generational degradation when repeatedly edited and saved	Lossy, but suitable for final distribution of photographic-type images because of smaller file size
TIFF (Tagged Image File Format)	Flexible image format, normally saves 8 or 16 bits per colour – red, green and blue – for a total of 24 or 48 bits	Can be lossy or lossless	Can be suitable, depending on the image itself
PNG (Portable Network Graphics)	The PNG file format supports true colour (16 million colours), whereas the GIF file format only allows 256 colours	PNG suits perfectly when the image has large areas of uniform colour. Lossless format, is best suited for editing pictures	It is suitable for large areas of uniform colour, therefore may be suitable for certain maps
GIF (Graphics Interchange Format)	Limited to an 8-bit palette, or 256 colours		Suitable for storing graphics with relatively few colours such as simple diagrams, shapes, logos and cartoon style images
BMP (Windows bitmap)	Colour depth: 1-24 bits	Files are typically not compressed, resulting in large files	Not suitable for final distribution of files because of large size

To summarize the information in the table, GIFF is not suitable since it is best for graphics and would result in big loss of image quality. BMP format is too large in size and least optimal. PNG and JPG formats are the best candidates, but it depends on the image of the map itself.

3.5.3 Persistent Data Store

One of the main advantages that the application is meant to offer is the ability to store the data in the mobile phone memory, which will eliminate the need for constant air-time to navigate the map.

One of the possible mechanisms to implement it is by utilizing the *persistent data store* support in MIDP (see chapter *Mobile Programming Technologies*). Real world applications produce data that needs to be saved, or persisted, and used subsequently by the same or another program. For this purpose a special mechanism for storing persistent data is dedicated in J2ME. Each MIDP-compliant device maintains a dedicated size of device memory for persistent application data storage, which depends on a specific device.

The idea behind the implantation of a persistent data store in the application is the ability to use the map with the shop displayed without a constant air-time. Once downloaded to the mobile phone the *Mobile Shopper Guide* application connects to the database and downloads the shop list with the related information that is stored on the device's persistent memory. This makes it comfortable to use since the only time user would like to connect to the database again is in case new shops emerge, in order to keep up-to-dated.

A risk behind this idea is that the persistent memory either will not be enough to save all of the shops, or the shop list will be limited. The size of memory depends on a certain phone model.

3.6 CONCLUSIONS

Requirements' gathering is the most important product development stage. In order the system to result in intended behaviour and satisfy the requirements, requirements have to be reasonable. For this purpose a feasibility study was run.

The basic principles of the feasibility study was described in this chapter, analyzing what is possible to implement, and what ideas should be given up. The feasibility study overviewed the use cases and concluded with the rough system architecture diagram and possible risks. Risks management requires a proper study, because in case the risks are real, they might stop the project from further development. To avoid this certain strategies must be engaged and back up plans must be used. Risks assessment was done in this chapter by pin pointing the risks, evaluating their severity and likelihood, offering a mitigating strategy and back up plans.

Having secured ourselves with the back up plans, requirements were gathered with respect to the results of the feasibility study. The chapter determined the requirements for the mobile platform – user requirements, functional, non-functional and technical ones. Next in this chapter an early GUI was designed and functionality behind it was described. The workflows in the early GUI helped to start the application structure analysis, like managing objects states and designing class structure. Optimization analysis for the best performance was done as well, keeping in mind that further optimization will to be done in the later development stage to further optimize the performance.

4. MOBILE PROGRAMMING TECHNOLOGIES

4.1 INTRODUCTION

There is a variety of possible programming platforms and technologies that can be engaged for the application development in a mobile phone. As mentioned fragmentally in previous chapters, one of the possible solutions is the J2ME package, for a couple of reasons. First of all, the shopper guide application is targeted for a quite variety of mobile phone models. Java package is suitable for almost every java-enabled phone, unlike most of the other technologies. Another point is the persistent data store mechanism for information storage within the phone memory that the J2ME offers. However, some other programming languages and technologies will be analyzed to make sure the right decision is made.

One of the most popular and wide used technologies, apart from Java, is Symbian, Brew, Andriod, Python and some others. The most significant criteria for the selection are: emulation available, the easiness to learn the language, suitable integrated development environments, platform deployment, graphical interface, runtime speed and the tool cost.

Table 6 lists all important factors and their evaluations for each of the technology in brief. *Learning curve* column is the evaluation of how difficult the technology is for the developer to construct the Hello world (simple program, text output) on the mobile platform. It also addresses the easiness of accessing advanced mobile features. This criterion is quite relative. *Emulator available* field addresses the emulators available based on integration with development emulation options. *Integrated development environments* show what environments, like Eclipse or Netbeans, are suitable. *Graphical Interface* shows the GUI capabilities and *Cross-platform deployment* evaluates the platform suitability. Each of the cross-fields is coloured in either of three colours – green, yellow or red, corresponding to high, medium and low suitability of the technology by certain factor. [9]

Table 6. Evaluation of the programming technologies

	J2ME	**Symbian**	**Brew**	**Android**	**Python**
Learning curve	Average	Difficult (unusual APIs, poor debugger support)	Difficult (but easier, than Symbian)	*unknown*	Excellent
Emulator available	Free Emulator, Sun Java Wireless	Free Emulator	No Emulator for the target ARM code	Free Emulator	Add-on to Nokia Emulator

	Toolkit				
Integrated development environments	Eclipse, Netbeans Mobility Pack	Many choices	Visual Studio 6.0, Visual Studio 2003 .net, Visual Studio 2005	Eclipse	Several, plug-ins for Eclipse
Graphical interface	2D, 3D graphics, many widgets, visual form-based GUI Builder	2D, 3D Graphics (newer phones), many Widgets, visual form-based GUI builder	2D graphics, 3D graphics (not available on all phones), limited widgets, no visual form-based GUI builder.	2D, 3D (OpenGL ES) Graphics, some widgets	2D Graphics access, some simple widgets
Cross-platform deployment	Average (device specific bugs needing separate builds)	Compile per target	Compile for the specific Brew version	*unknown*	Available natively only on Nokia Series60, but there are ports to other mobile platforms
Runtime speed	Average due to Java bytecode	Best (Compiled language)	Best (Compiled language)	Average due to Java bytecode	Below average due to interpreted language
Development tool cost	Free	Varies (free tools available)	Fees Required for Brew App Certification	Free	Free

From the table it is visually obvious which technologies might be suitable. The most preferable technologies have green-coloured factors. However, some factors being less than excellent (colour yellow) may be acceptable for not being crucially important. Apart from J2ME, having all criteria acceptable, Symbian might be a possible solution, let alone the learning factor. Also, Brew is acceptable, if we ignore the fees. Python might not be a good solution since the runtime is below average and we aim to provide an application that is optimal among other things in performance. Android is not so popular and very few information about it is available.

4.2 SYMBIAN OS

Symbian OS was built to emphasize three basic things – the integrity and security of user data, runtime speed, and working as if all resources are scarce. There is a strong emphasis on conserving

resources, using Symbian-specific programming idioms. There are similar techniques for conserving disk space. Furthermore, all Symbian OS programming is event-based, and the CPU is switched off when applications are not directly dealing with an event.

Good points for the Symbian OS are, in short:

- The Symbian OS from developers point of view is suitable for handheld devices with limited resources;
- The Symbian OS from user's point of view is a consistent user interface, comprehensive set of personal information management applications, extensible by 3rd party applications;
- It supports multitasking, multi-threading, memory protection, it is event-based;
- Special features involve conserving memory, reliability, CPU is switched off when applications are not dealing with events. [28]

Symbian compatible devices:

- Series 60 (smartphones with numeric keypad, "phone centric"), e.g. Nokia 6600
- Series 80 (organizers with full keyboard, "information centric"), e.g. Nokia 9300
- UIQ (smartphones with pen input, "information centric"), e.g. Sony Ericsson P910 [28]

Several developer tools can be used on a Symbian OS, such as:

- Microsoft Visual Studio – C++ IDE
- Metrowerks Code Warrior – C++ IDE
- Abld
- Makesis and others

To start Symbian development, a developer needs:

- a PC running Windows NT or better
- an SDK (S60 or UIQ – Symbian based platforms)
- a C++ development IDE supported by the SDK
- The ability to deploy code onto the phone (for example, Bluetooth)

Not all phones with Symbian OSs support MIDP 2.0. MIDP 2.0 is generally supported by Symbian OS v7.0 and higher.

4.3 BREW

Brew (Binary Runtime Environment for Wireless) is an application development platform created by Qualcomm for mobile phones, in September 2001. Brew runs between the application and the

wireless device's chip operating system so as to enable a programmer to develop applications without needing to code for system interface or understand wireless applications. [17]

What is specific about Brew, unlike the Java ME platform, where any developer can upload and execute software on any supported handset, Brew applications must be *digitally signed*. Because Brew gives complete control over the handset hardware, only content providers or authenticated Brew developers have the tools necessary to create a digital signature. Furthermore, developer-signed applications can only execute on *test-enabled* handsets, which in our case is not a perfect solution. Once the application has been developed and internally tested, it must be submitted to NSTL for TRUE BREW Testing at significant additional cost. Before submitting for testing, the application needs to be signed. In March 2006, the least expensive digital signature for testing costs *400 USD* and is limited to 100 application submissions. After the application passes all tests, it may be offered to a mobile operator (content provider) to be accessible for download to general handsets. The application is then signed by the content provider, to allow its execution on any supported Brew handset. However, after all these hurdles have been cleared, there is still a high risk that carriers will reject the application as insufficiently profitable, or as a competitor to one of their own applications…

A thing against the *Learning curve* factor (Table 6), a Brew application consists of several files. It must contain several necessary components, otherwise it will be automatically deleted on reboot. These are: *name.mif* file which describes the application, the features it uses and permissions requested, a *name.mod* file which is the actual compiled binary, *name.bar* containing string and image resources if required, and a *name.sig* which is the application digital signature. Applications which do not have or have an invalid or expired digital signature are automatically deleted on reboot.

There are however, some advantages of Brew, compared to J2ME, like:

- Graphics tricks are easier, especially with Brew 2.0 and direct access to the screen buffer. This feature, however, is not significant for the *Mobile Shopper Guide* idea.
- The development environment is standardized, comes with free tools, and contains a well integrated emulator that properly reflects the behaviour of the actual device (J2ME emulators differ in reliability, performance, integration, and ease of use).
- Once a Brew application is working in the emulator there is a high probability it will work on all Brew devices. However, different phones may implement some Brew API aspects differently and debugging such issues can be difficult since the emulator tends to be more generic.

- Brew applications perform much faster. J2ME performance might be quite poor, in comparison, when attempting time critical tasks. As for the idea of *Mobile Shopper Guide*, not time critical tasks are to be completed. [17]

Things that might not be preferable for using Brew as application programming technology, compared to J2ME, include:

- In J2ME, the entire source file and resources are compressed by default. To compress resources with Brew is different. It can only be compressed if you implement your own method or buy a commercial solution.
- The Brew developer community is fairly small and limited to Qualcomm's boards and web sites. There are not many books available either.

These are the main drawbacks that stop from the idea of using Brew to support the *Mobile Shopper Guide* idea.

4.4 J2ME PACKAGE

As mentioned before and noticed in the table of technology features comparison, J2ME is the best solution for the shopper guide. Apart from the persistent data store mechanism provided and a variety of phone models that support java (for the entire supported device list refer to [29]), there are several other good points supporting this solution:

- Network connection (usually http)
- Bluetooth support
- GPS support (ability to retrieve satellite coordinates, this can be valuable for future application feature extension)
- Portability

Connection feature is necessary to get the updates from the database. Bluetooth is significant for the application download to the mobile phone. Portability in Java ME allows mobile phone applications to run on diverse devices – without recompilation. J2ME application is delivered as a group compiled class and data files stored inside a JAR file. Additionally, a descriptor file – a "JAD' file – is generally provided to describe the contents of the JAR.

What is more, Java2 Micro Edition (J2ME) offers a special environment for creating platform independent applications on mobile phones and other handheld devices. J2ME is divided into two parts: *configurations* and *profiles*. The configurations are specified by the memory size and computing power of the device, whereas the different profiles are specified by the user interface. The *Mobile Information Device Profile* (MIDP) is targeted for the mobile phones and is widely

supported by all the major mobile phone manufacturers. Devices implement a *profile*. The most common is the *Mobile Information Device Profile* (MIDP) aimed at mobile devices, such as cell phones, and it defines a set of user interface (UI) APIs designed for wireless devices. Profiles are subsets of configurations. There are two of them: the *Connected Limited Device Configuration* (CLDC) and the *Connected Device Configuration* (CDC).

The Connected Limited Device Configuration supports personal mobile devices and comprises a less powerful class of devices than the CDC configuration supports. The CLDC specification defines a platform for devices with the characteristics:

- 160 to 512 KB total memory available for the Java platform
- 16-bit or 32-bit processor
- low power consumption, often battery powered
- intermittent network connectivity (often wireless) with potentially limited bandwidth [6]

A configuration provides the most basic set of libraries and virtual-machine features that must be present in each implementation of a J2ME environment. MIDP is designed for cell phones. Applications written for this profile are called *MIDlets*. We will target MIDP 2.0., after reviewing and comparing MIDP 1.0 with 2.0 and perhaps 3.0 versions. MIDP 1.0 is too limited, it has no active rendering APIs, no support for direct access to image pixels, has no support for full screen mode. Also specifications are not always clear, which lead to considering MIDP 2.0 version. MIDP 2.0 delivers an enhanced user interface, greater connectivity "over-the-air". Also, MIDP 2.0 provides a more flexible layout for greater application portability. MIDP 3.0 specifies the 3rd generation Mobile Information Device Profile, improving interoperability across devices. A key goal of MIDP 3.0 is backward compatibility with MIDP 2.0 content. [30, 18]

MIDP applications are Java applications, which means that Java 2 Platform, Standard Edition (J2SE) and its tools are needed to build them. J2ME Wireless Toolkit (WTK) is a set of tools that provides application developers with the emulation environment, documentation and examples needed to develop Java technology applications targeted for CLDC/MIDP compliant mobile phones. As for emulation, the WTK can be used. It does not include a facility for editing code, but it includes everything else that is required – means for arranging, compiling, verifying, signing, packaging, deploying and testing Java ME applications. J2ME and development tools support device emulators by device manufacturers and therefore testing applications on different platforms becomes easier. However, the performance and features of the emulators may be different from the real device, applications should also be tested on the actual devices, to see whether or not it works correctly also.

One of other factors for choosing J2ME is some experience on Java already attained. It is easy to develop a Java ME application if one is already experienced Java. In fact, Java ME development is

a cut-down version of Java SE (Java Platform, Standard Edition). Many of the classes and interfaces are the same as the ones in Java SE. It can be argued that developing a Java ME application is even simpler than developing a Java SE application.

4.4.1 MIDP Application Program Structure

It is necessary to understand the basic MIDP programming model and the abstractions defined by the UI components in order to build user interfaces. But the most important is to understand the MIDlet execution lifecycle.

When the application is first launched, it starts the Virtual Machine (VM) and instructs it to instantiate the MIDlet class. The *Application Management Software* (AMS) then calls the *startApp()* method, which is responsible for the initialization. The startApp() method is always called as the entry point for your MIDlet. But if J2ME applications are real Java applications, there must be a main method that is the real entry point used by the VM to start the process of executing the application. This method exists and it is a part of the MIDP implementation, and, typically, the AMS software calls it. The details are implementation dependent. In Sun's J2ME Wireless Toolkit, the class *com.sun.midp.Main* defines the main() method. The startApp() method creates an object called a *form* and passes a string to the constructor that represents the form's title. A form is an instance of the class *javax.microedition.lcdui.Form*, which is a kind of screen that you can see on the display.

MIDlets have different states during their lifetime. The MIDP specification defines the MIDlet state transition model. See Figure 4 for the MIDlet state transition diagram. The startApp(), pauseApp(), and destroyApp() methods allow a MIDlet to change its state.

Table 7. MIDlet states

MIDlet state	**Description**
Paused	MIDlet's execution is stopped until it transitions to the active state.
Active	MIDlet is either ready to run or running. The thread that controls the MIDlet might not be in the run state, but MIDlet is still active.
Destroyed	The MIDlet isn't running and can no longer transition to other states.

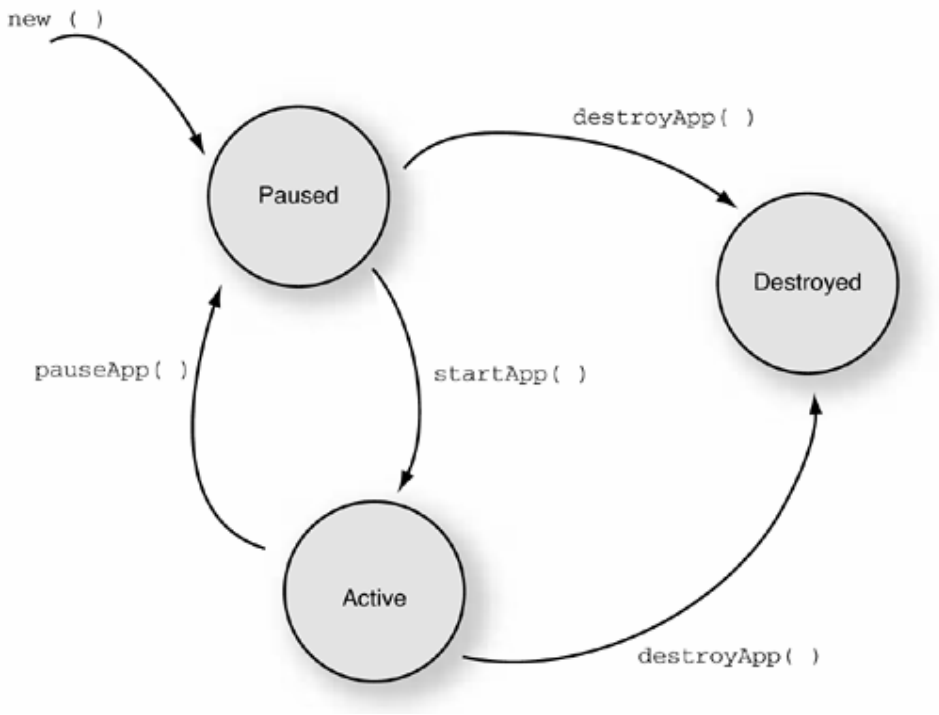


Figure 4. MIDlet state transition

Refer to Table 7 for the description of MIDlet states.

Talking abut the user interface, MIDP UI components are defined in the *javax.microedition.lcdui* package. The Display object represents the physical display of the device's screen. It is a *displayable component* that can have a visual representation. A displayable component in MIDP is a *top-level* UI component, meaning they can be independently displayed by a MIDlet (they don't need to be contained inside any other component). A MIDP application can display only one top-level component at any given time.

When the AMS launches a MIDlet, it does the following:

- Instantiates the Display class
- Associates the Display instance with your MIDlet instance

The program never creates the Display object; the MIDP implementation does. MIDlet creates only the UI components – instances of the concrete subclasses of Displayable or Item that will be displayed on the screen throughout your MIDlet's lifetime.

Three primary objects interact: the MIDlet instance, the Display instance created by the AMS, and the Displayable component that you want to appear on the screen. See Figure 5 for a UML diagram of the relationship between these objects. Figure 5 shows inheritance diagram of all the classes in the javax.microedition.lcdui package.

The main components of a MIDlet's structure are the MIDlet instance, a Display instance, and one or more Displayable widgets, which are UI components. MIDlet objects are associated with a

Display object. MIDlets create Displayable widgets, which are UI components, and request that they be displayed on the device's screen. The display manages the device's screen and the visibility of the UI widgets. The abstract Screen class is the first of two main types that categorize all Displayable objects. The Screen class is the central display abstraction. The Form class is a concrete subclass of Screen. Only one Screen is visible at any moment in a MIDlet's life.

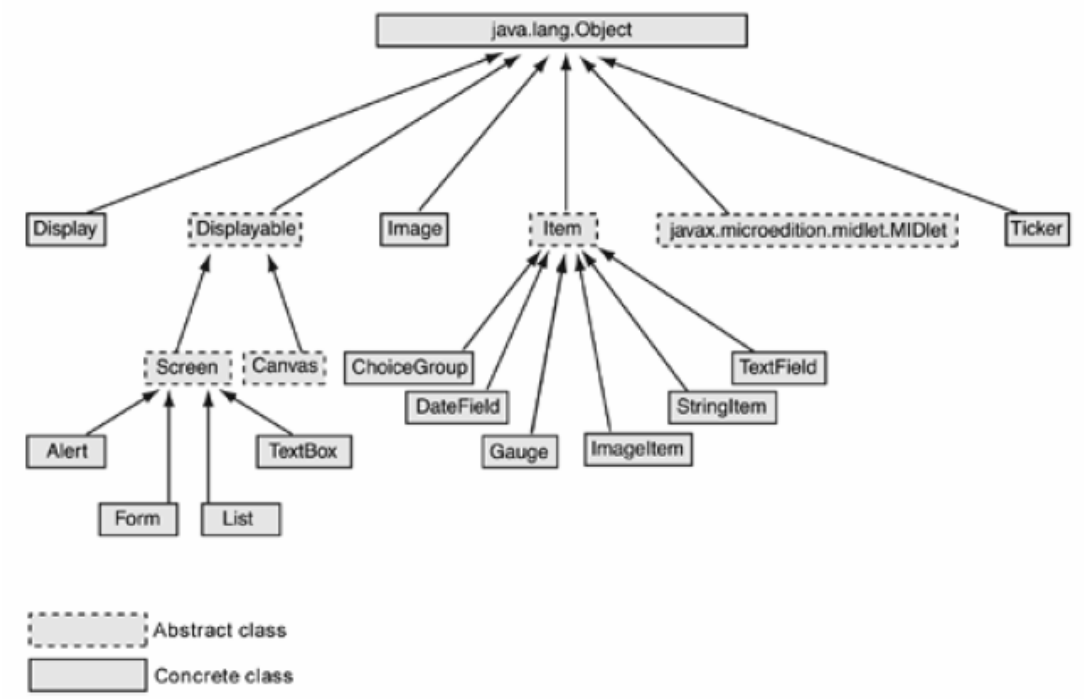


Figure 5. The inheritance diagram of the MIDP UI components

4.4.2 Persistent Storage Support in MIDP

The MIDP supports persistence application data store through its *Record Management System* (RMS), the *javax.microedition.rms* package. Each MIDP-compliant device maintains a dedicated area of device memory for persistent application data storage, were the data stored persists across multiple invocations of the applications using it. Both the physical location and the size of the data store are dependent on a particular device model. The RMS API abstracts the device dependent storage details and access to those details and provides a uniform mechanism to create, destroy, and modify the data. This ensures portability of MIDlets to different devices.

The RMS supports the creation and management of multiple record stores, shown in Figure 6. A *record store* is a database whose central abstraction is the *record*. Each record store consists of zero or more records. A record store name is case sensitive and can consist of a maximum of 32 Unicode characters. A record store is created by MIDlet.

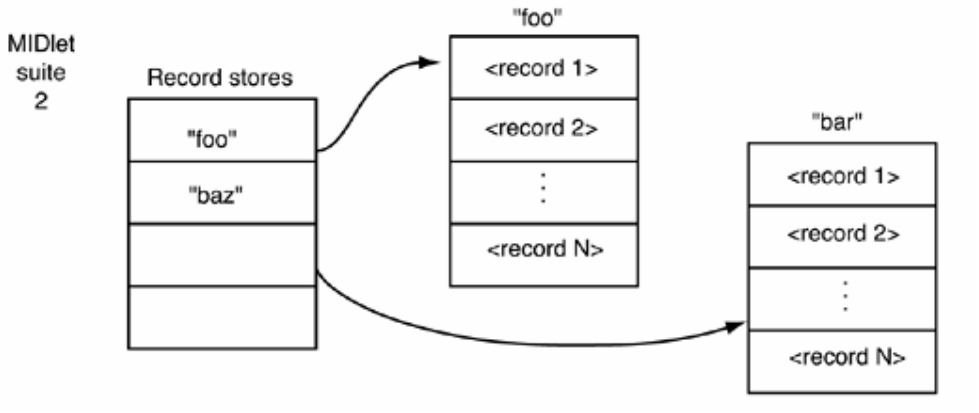


Figure 6. The RMS records

The RMS defines the following conceptual operations on an individual record store:

- Add a record
- Delete a record
- Change a record
- Retrieve a record
- Enumerate all records

A record is a byte array of type byte. The RMS doesn't support the definition or formatting of fields within a record. Therefore, the application must define the data elements within a record and their format. And the reader of a record, therefore, must be aware of the format that was used to write the record.

4.5 TOOL SELECTION

This chapter reviewed several mobile programming technologies and analyzed their cons and pros and suitability for the current background, i.e. programming knowledge of the project participants, tool cost, emulations available, suitable integrated development environments, graphical interface, runtime speed and some other factors. Three most suitable solutions that corresponded factors relevancy, were chosen and analyzed. Having compared them together, Java2 Mobily Pack was chosen as further application development tool. Greatest advantages that made impact for the decision are free tool cost, java programming experience already acquired, portability throughout different platforms, graphical interface, a well supported persistent data storage mechanism and free emulation possibilities.

5. SYSTEM DEVELOPMENT

Detailed requirements specification is the key to produce software in accordance with planned functionality and customer needs. Another important aspect is application testing on as many different phone models as possible. Apart form testing with emulators several real devices shall be tested. Application development process is iterative and an early testing must be performed by the developers before it is introduced to the user. This is done to avoid design errors, flaws and to improve usability because in the late stage those errors may even require reorganization of the whole system.

The *System Development* chapter covers system design issues sequentially: creating initial state diagrams, designing class hierarchy, early testing by *cognitive walkthrough*, usability and final acceptance testing. Test results in Appendixes summarize flaws detected by which redesign was done or future ideas were noted.

5.1 CLASS ANALYSIS AND INITIAL HIERARCHY

Five basic classes in the class diagram has to be implemented: *System, GUI, Map, PersistentStorage, and ServletConnection*. Class System would be responsible for objects destruction and garbage collection, class GUI – for graphical user interface, Map would take care of the different map views and objects displayed, and PersistentStorage and ServletConnection classes would control connection to the database and the data storage on the mobile phone memory.

A more refined state diagram was built in later stages having the initial class hierarchy as the basis. Figure 7 is useful for knowing how the objects involved change their states with response to the events and time. The state diagram below shows the start and the end points of a sequence of state changes, denoted by black circles. The state diagram is divided into two big areas. On the right you can see the state of the objects connected with the main menu, and on the left – states and objects of the overview map and the detailed map. Some commands that the user will have to press to get to other objects (like clicking the arrows or the 2,4,6,8 keys to get to the state “move the pointer”) are shown in the diagram.

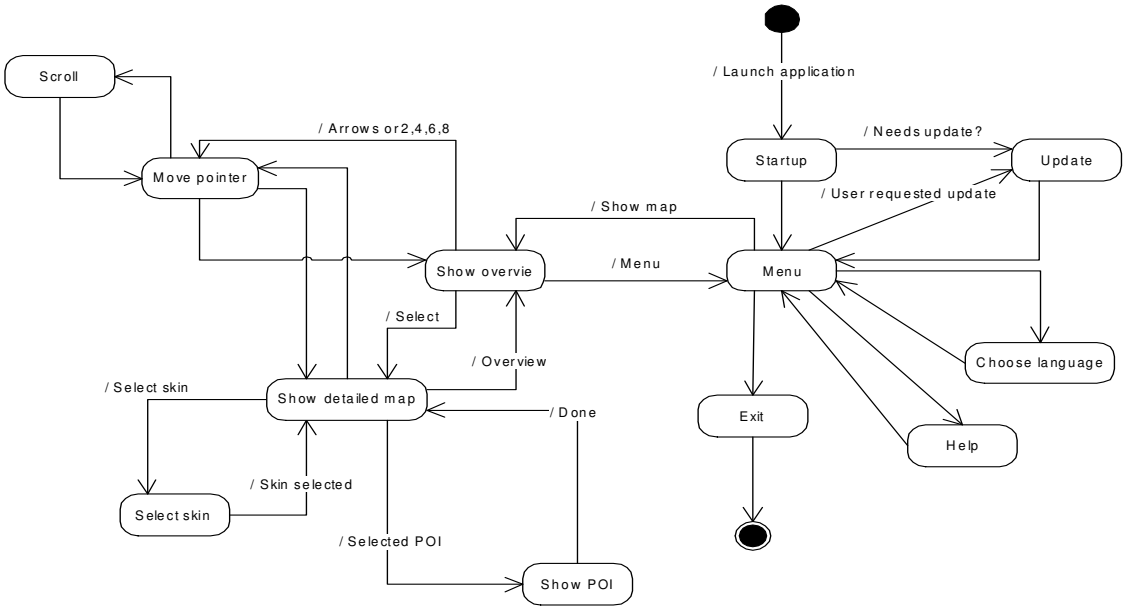


Figure 7. Initial state diagram of the mobile platform

In essence, the application consists of four main *packages*: *gui*, *connection*, *map* and *storage* (see Figure 8).

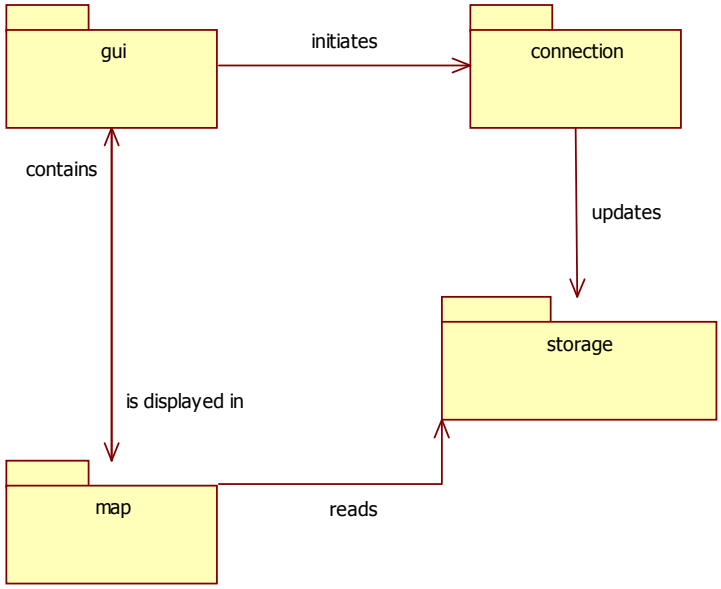


Figure 8. Main packages and their relations

Figure 8 shows the main packages only. Digging deeper, on the whole the class structure resulted in having total of 8 packages:

- *connection*
- *debug*

- *gui*
- *map*
- *servlet*
- *storage*
- *struct*
- *system*

The *connection* package includes classes that are responsible for establishing the connection between the J2ME application and the database server. The ServletConnector class is responsible for this. *Debug* package is for the debugging purpose only. It consists of simple classes with static methods to perform debugging in order to track the object states. Package *gui* is the main one, it includes the startApp(). This package includes a MIDlet class and is responsible for starting, pausing and destroying the application. It also handles user actions when the map is visible. Package *map* holds classes for managing the map display. It also holds the *singleton* class which is used for creating, managing and destroying different maps. *Servlet* package involves the servlet that is responsible for the connection to the database. The MIDlet in the connection package calls the servlet (via http) that is put on the server and which connects to the database and fetches the data requested by the calling application. The servlet on the apache server is configured not to interact with the database before a valid username and password from the mobile application is received. These values are predefined in the software and transmitted in the background without the need of user-interaction. When the servlet is successfully connected to the database, it starts to send SQL-queries and stores the responses in a text-string which is finally sent back to the mobile phone. The phone in turn stores the data in its persistent memory. *Storage* includes a static class for reading and writing to the device's persistent memory to store the shops, their coordinates. This class takes the advantage of the RMS, supported by the MIDlet. Another static class exists in the package which deals with a non-persistent representation of the points of interest for fast access when scrolling and repainting the map. *struct* holds all basic structure class, used within the application, like Box (having the coordinate of the top left corner and width and height), Coordinate, POI (representing a complete point of interest). Classes within package *system* are responsible for storing the constants needed within the execution of the application, a class for garbage collection to free the memory, and a class to keep the state of the application, like language selected or last updated (the states, however, are not kept when the application is exit).

5.2 SYSTEM DEVELOPMENT AND FUNCTIONALITY

Later, when the development started according the low-fidelity GUI, a more refined class hierarchy was developed. It corresponds to the high fidelity state diagram in Figure 9. Basic functionality according the Figure is:

1. *Retrieve updates from the database*. This first thing that is offered when the application is first launched. User can choose to connect to the database for updates form within any state of the application, from the main menu.
2. *Show overview map*. This is the default map – the zoomed out view of the map. User can navigate the overview map using the softkeys or digit buttons.
3. *Zoom in*. Form the overview map user can zoom in to the selected area. The rectangular indicating the area is attached to the navigation cursor which can be zoomed closer for the detailed view of the map. By default all types of shops are listed on the zoomed out map.
4. *Choose shop type in detailed or overview map*. While in the detailed map user can select a shop type to display only shops of that type. Currently, these shop types are hard-coded: Restaurants, Shoes, Clothes, Children shops and Cosmetics.
5. *Show all shops of certain type*. After choosing a certain shop type, only selected shops are displayed. Then user can either navigate on the icon on the map, or choose to get the list all shops of a certain type.
6. *Select a shop from the list, read info and show it on the map*. Form the shops list user can select a certain shop and read information about it. Then user can select to show the shop on the map. This will not open the location on the map where the shop resides directly, but rather the navigation arrows will indicate the direction to follow. This decision was made so as not to confuse the user by directly opening the part of the map without an idea of how to get there.
7. *Navigate on an object on the map and get info*. Shops on the map are represented by little red circles. By navigating with the cursor in them a pop-up info will show saying the name of the shop. User can select form the main to read information about that shop.
8. *Help*. User can read about the application by selecting Help from within any state of the application.
9. *Exit*. User can exit from within any state of the application.

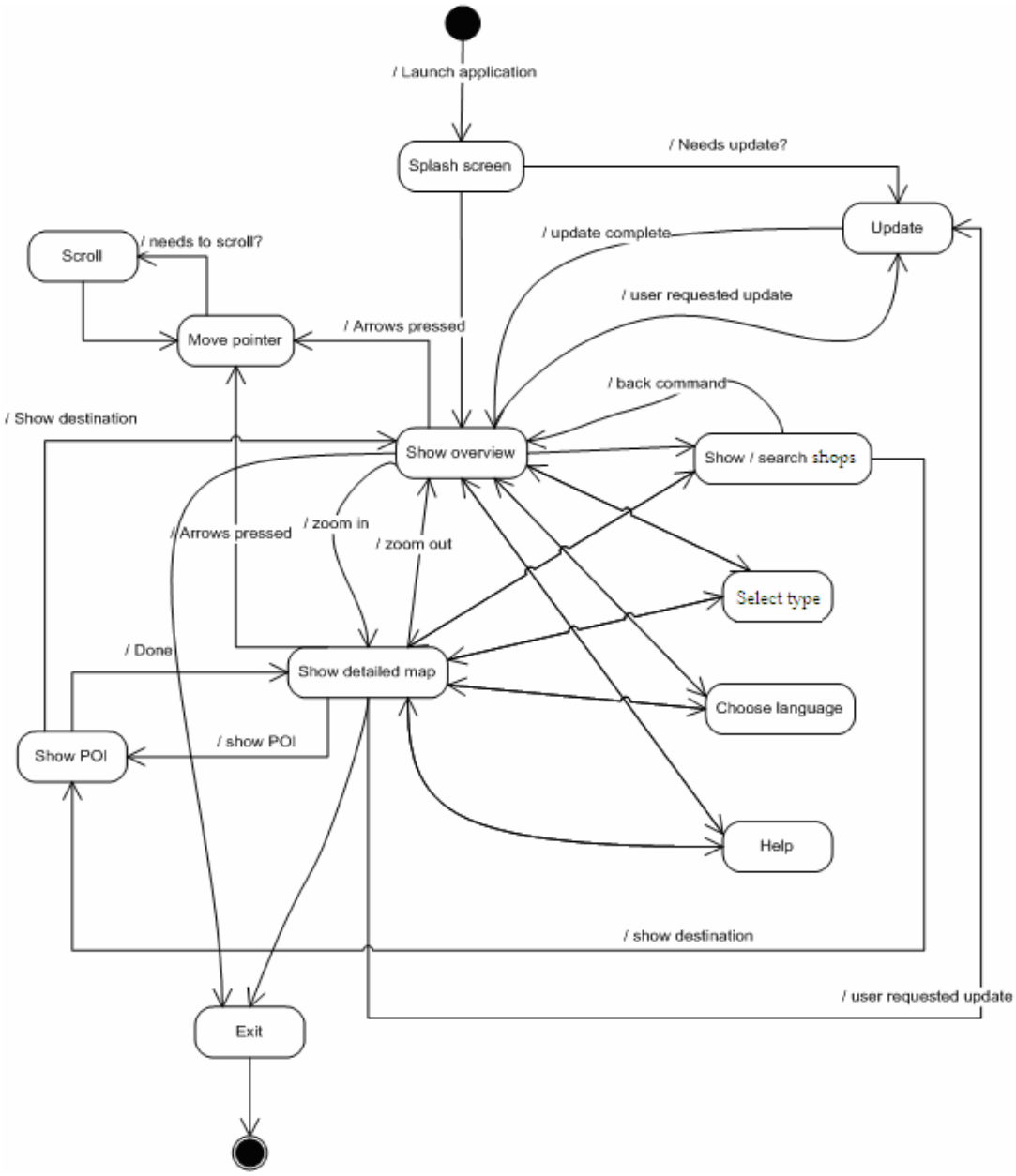


Figure 9. High-fidelity state GUI state diagram

5.3 TECHNICAL IMPLEMENTATION DECISIONS

Some technical considerations were also considered. First of all, the connection to the database. For the connection from a MIDlet running on a mobile phone to the database, the following components were required:

- A *thread* managing the requests for the connection to the database. It sends a request to connect, which is then passed to the servlet on the Apache Tomcat Server, and holds the URL to the database via the http protocol. It also stores the user name and password

required for connection. This class is responsible for putting the retrieved records to the persistent data storage.

- A class that extending *HttpServlet* and managing the connection to the database. The class – servlet – is put on the Database Server. It holds SQL scripts and runs them on the database once connected and retrieves the query results in a form of text strings. They are send back to the calling MIDlet.
- *Tomcat Server*, where the servlet is put. Tomcat server has to be properly configured to manage the http requests.
- *SQL Server database* has to be running and configured to listen to certain protocols.
- A user portal needs to provide Help for the user. User can read help locally on the phone by selecting appropriate menu option. Also a link to wap version of user portal is available to more extensive information. For this, the server managing *User portal* has to be accessible.
- The user portal is also used for *JAR and JAD files download* directly to the mobile phone. Another possibility is to download the JAR and JAD files to the PC and then transfer to the mobile device using *Bluetooth* technology.

5.4 TESTING AND FURTHER OPTIMIZATION

5.4.1 Raster vs Vector Map Image

As already mentioned, raster images for the map are preferred to vector images. The optimal solutions are PNG and JPG raster formats, however, it depends on the picture of the map itself – quality and size may differ if the map has uniform colour areas compared to detailed and different-colours setting. A simple test calculating the size and quality for a certain format was run. Table 6 lists all formats that were used and the application and image sizes. *Quality* in this test is a four scale measure: Very good-Good-Poor-Very poor.

Table 6. Raster format comparison

Format	Quality	Total image size	Application (jar) size
JPG	Good	608,8 Kb	687 Kb
GIF	Poor	894,7 Kb	974 Kb
BMP	Very good	12731 Kb	4,08Mb
PNG	Very good	4287 Kb	4,65 Mb
TIFF	Good	516,12 Kb	5,74 Mb

Quality is the best for BMP and PNG formats, however, these formats take too much space. JPG format takes the least space while at the same time retaining good enough quality. It is clear enough that the less space the application takes, the quicker is the navigation and performance. To make sure JPG format was the best decision in terms of quality losses, and formats of better image quality were neglected meaningfully, another simple test was run. This test was meant for response time comparison with respect to different formats (and therefore the different application sizes). Table 7 lists four user actions that are easy to repeat and calculate the response time equally.

Table 7. Response time comparison

User action	JPG (s:ms)	GIF	BMP/ TIFF	PNG
1. Load the map	00:00	00:00	Doesnot load	00:00
2. Update the map with new shop list	06:08	06:17	06:08	06:82
3. Navigate from one side of the map to the other (overview)	06:61	07:04	05:44	07:88
4. Navigate from one side of the map to the other (detailed)	24:80	27:01	13:60	29:47

BMP and TIFF formats take too much memory space and, therefore, neglected. The quick response time is only because no images are loaded. GIFF response time is similar to JPGs, however the quality of the image is poor.

Naming Images

Also for quicker access, image filenames are stored in the form 1XXX1YYY.jpg, where X and Y is the title number starting from 0. In this case, no specific data structure is needed to store image names, only simple function returning a correct image name based on a simple algorithm. For the current project only about 64 different pictures for the detailed map are needed, so the time savings are not so prominent. However, whole city map would required a lot more of the images and storing the names would result in a non-efficient usage of memory and, hence, time lags.

5.4.2 Utilizing Persistent Data Storage

For the data to be stored on the persistent mobile phone memory, a RecordStore object is created from the MIDlet. The question is does the memory allow to store a lot of POIs? A test was run to see how many of the POIs could different mobile phones store in the flash memory (where you store data permanently). It appeared that record size varied depending on the different mobile

phones. For the Nokia mobile, the record manager system size was only 63 Kb. This could cause problems if one POI had size bigger than 1KB. We discovered that 500 characters of information were probably enough for one POI, at most it could take about 0.3Kb. This means that for Nokia case we only can store about 210 POIs. This could be enough for a city like Vilnius, especially if only the centre part is of interest. But in the future, for a bigger city, or more suburbs included, you might need to find another memory storage if it needs to work on as many phones as possible.

Volatile data

Another data structure is used for a non-persistent representation of the points of interest for fast access when navigating and repainting the map. A non-persistent representation means a dynamic representation of the shops on the map while navigating, zooming in and out. This makes the navigation pointer aware of the points of interest when the pointer is over and which direction to head for when looking for a specific object. To manage this kind of data a separate class was used.

5.5 USABILITY TESTING

Usability testing is a technique used to evaluate a product by testing it on users. It measures the usability, or ease of use. During the usability testing, the aim is to observe people using the product to discover errors and areas of improvement. The application was first tested by the project members before it gets to the en-users. For this purpose the *cognitive walkthrough* [7] was run. One of the characteristics of the cognitive walkthrough is to use it when a system developer wants to establish how easy a system is to learn; you learn through exploration [7]. A cognitive walkthrough starts with a task analysis that specifies the sequence of steps or actions required by a user to accomplish a task, and the system responses to those actions. The participants ask themselves a set of questions at each step. Data is gathered during the walkthrough, and afterwards a report of potential issues is compiled. Finally the software is redesigned to address the issues identified. Kjeldskov et al. (2005) discussed and tried tests with both expert evaluators and end-user evaluators. Kjeldskov et al. (2005) recognize that there could be some problem with not using the end-users but they also say that the expert evaluators are perhaps more useful to use when testing in an early phase. [8]

To do this test we first came up with a task that the evaluators should try to perform on the system. This task should be a representative task that most users will want to do. Our task was: “Download and update the shop list and show information about shoe shops that resides in the centre of Vilnius and then show the other show shops in the area left to the centre”. Then a written list of the actions needed to complete this task was written down. When doing this test the evaluator went through all the actions and with every action completed the user answered four questions. These four questions

and all the actions, and a more fully description of the test, is described in *Appendix C: Cognitive Walkthrough Test, Appendix D: The Test Subjects, Appendix E: Failure Document.*

Quite a few flaws were found in the design; they were mostly design issues that the user would have problems understanding. The evaluators experienced problems with how some words/functions were phrased. They, for example, recommended that the "update" has to clearly state what the user is updating and that the update is successful. There was also some controversy on how the user would know where certain places reside in Vilnius. This could be fixed by drawing out landmarks or something similar on the overview map, and perhaps also on the detailed map.

Also several navigation issues were discovered and we got some feedback on how we could solve these problems. There was a discussion, that didn't have anything to do with the questions at hand, if a main menu was really necessary. It would perhaps be easier for the user if the overview map was shown directly with a pop-up menu instead.

We also discussed how you would scroll in the detailed map, i.e. how fast and what feedback the user should get. All the evaluators agreed that turning off the 'pointer' was a bad idea.

5.6 FINAL TESTING AND VALIDATION

The software testing, like the GUI test is important to verify if the application is meeting the specifications, if it does, then what it is required to do (according to the requirements) and if it is usable. The application was tested on three to four different kinds of mobiles phones, either a SonyEricsson or a Nokia mobile.

The first test was run to see if the application could get a connection with the database for all of the shop types. These were the main requirements to test and get a feedback on:

Mobile platform

Technical requirements

1.3 A java enabled phone with a colour screen and enough free memory

Functional

1.2 Establish a **connection with the database**

The connection to the database was established successfully provided the SQL Server was running and the servlet managing the connection to the database was put on Tomcat server. We also tested how much the RMS (record management system) a phone could store, i.e. how many of these POIs our different mobile phones could store in the persistent memory. We found out that the RMS size you could use was various depending on the different mobile phones used. On, say, Nokia mobile phone, the RMS size was only 63 Kb. This could be a problem if one POI had a size that was bigger than 1KB. We discovered that 500 characters of information were probably enough for one POI, by

putting up a restaurant's week menu, the largest information we could think of, and this information totally would take about 0.5Kb. This means that, if we look at the limit that our Nokia test mobile has, we only can store about 120 POI in the RMS. The RMS test was important to see how much space the point of interests would take on the mobile phone.

The next thing to check is whether the map was too big for some mobiles and if it was visible enough on the different models. We had to consider screen size and quality on the different mobiles. This was a very iterative procedure. The mobile phones have different resolution and they also have different capacity when it comes to DPI (dot per inches). Therefore one mobile could display a bigger part of the map than another model.

We found that a 128x160 resolution screen was the minimum requirement that the user should have to display the map on their mobile. We also found out that in order to get it to run on, for example, the Nokia phone we had to resize the map and also change the colour depth to a much lower value. In the end it seems that we would have to change our max mobile application size to 1Mb.

5.7 ACCEPTANCE TESTING

Acceptance testing involves the *end users*. We created five modules, and what steps every module should be tested on for the acceptance on the application from the end-users. With the five modules we also made tasks and instructions that the testers should complete before moving to the steps that every module have.

- Module 1: Update the application

  - The user should connect to the database and update the application.
- Module 2: Choose shop type

  - The user should choose a shop type, e.g. "Restaurants".
- Module 3: Navigate the map

  - The user should navigate to "Delano" where they need to find the fictive restaurant that's located at the street "Stepono".
- Module 4: Information about the object

  - The user should find information about the object, the restaurant and find information about the menu.
- Module 5: Exit the application

  - The user should exit the application

Every module has seven steps that the tester should try to answer before moving on to the next module. The seven (questions) is a combination of usability steps and functionality steps. If they think that a step have been fulfilled they mark it in the table and set the date and their initials. If they feel that one of the steps in a module is incorrect they write a failure report for that step e.g.

they write what the problem is with this step in that specific module. Then we ask the tester basic questions about the application, for example, how they think it works in general and what they like and don't like about the idea. *Appendix F: The Acceptance Test* shows what steps we made for each module and also what final questions were the end-users asked. The end-users were native and we didn't use any tourists during our test.

We got quite similar answers from the end-users. Overall, the users seem to be happy with the mobile application and thought that it was pretty easy to use and they also saw potential in the application. They thought that it was a bit unfortunate that it looked so different on different mobile phones and that it was missing a GPS (global positioning system) function. They also thought that it would be nice if the application had a search function (so you could search for a specific area or street). Other than that we got both positive and negative feedback about the design and the functions (although there was mostly positive feedback).

Between our test phones, a Nokia 6111 and a Sony Ericsson K750i, the feedback was about the design and why it looked so different on different phones. The Nokia phone has a small screen so it means that you can't see as much of the map as you can with the Sony Ericsson mobile. Sony Ericsson was more useful because it showed a larger part of the map (which meant that you didn't have to scroll as much). Someone recommended that this could have been partially solved if we could implement another overview screen so that you didn't have to scroll that much. We didn't have time to implement this function and we thought about it before, but we decided that it could be cumbersome for the user to have to press two times to get to the detailed map. We also got response on the differences between the alignments of the buttons depending on the mobile phone. This is something that we can't really change; we would have to do two versions (one Nokia version and one Sony Ericsson version).

Other design issue's that was brought up during our test, that wasn't tied to a specific phone, was how illogical some of the menu text was. The end-users weren't sure of, for example, what the "show" command in the menu meant. One of the users thought that it would take you back to the map (it worked like this when he used the function before). The "show" command was supposed to show info about an object, but it only worked when the marker was over the object. The user discovered this by mistake and he thought that it didn't make much sense. We did have a chance to change it and the "show" command is now visible only when the user is over an object (this means that the user can't zoom out when they are over an object).

The end-users seemed to think that the functions worked correctly and that it shouldn't be a problem to use the menu after a while, or as one user puts it; "every user that has some experience with their own phone's menu system should be able to use it". Our application crashed one time when the user was navigating the map with the marker. We recognized this problem, which was

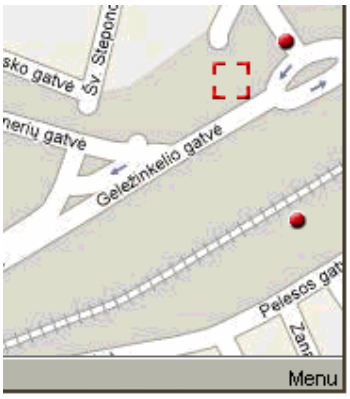
trigged when the user pressed two buttons at the same time just before they were scroll the map, and we are confident that we have fixed it. We are, though, aware that the application still can freeze sometimes and this seems to be limited to what phones the application is using. The test showed that our functions worked, even if we have some design issues. One of the users said that this application would be more useful for a tourist, when finding a specific restaurant, than "asking a citizen that lives in the city".

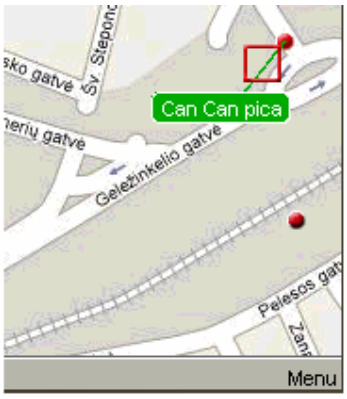
5.8 Design Decisions for Graphical User Interface and Soft Keys

Due to the limited input possibilities of a small handheld device all high-level user input is added as commands. These commands are then placed appropriately by the device manufacturers' implementation of java. This solution has both advantages and disadvantages. The approach is necessary to guarantee that applications remain as portable as possible between different devices. On the down side is the fact that developers has less control over the user interface.

Soft key assignment is one of the most prominent interface problems discovered during our project. Different hardware devices assign commands to different soft keys. They sometimes place them in a menu and sometimes not. We found it too time consuming to develop specific versions for different hardware, therefore our implementation is not perfect regarding the design of user input, but it is, we believe as close as we could get producing a single portable version. To further clarify the problems surrounding input design a few examples from the design process will be provided.

The actions most important for the user should be the actions that require the least amount of work





to perform [2]. Since most phones have only two soft keys we eventually faced the problem of which action that was the most important for the user when zoomed in, to display information about a point of interest, or to be able to zoom out effortlessly? Only one of the commands could be displayed as a soft key while the other was placed in a menu located under the other soft key. The solution is somewhat of a compromise. The soft key is assigned to different commands depending on context. When the pointer is over a point of interest the natural thing to want to do, is to display its information, when not the user probably has to zoom out as his/ her highest priority.

Navigation

The map is navigated using a pointer that detects when it is over a point of interest or when near the boundaries of the screen or the map. In order to make the user aware of possible actions the pointer gives contextual feedback described below. We wanted to give positive feedback back to the user, e.g. we wanted the user to know that an initiated action would not go unnoticed and we therefore made the pointer change according to different states. Feedback is very important, as the user doesn't know what you, as a designer know, they only know what you tell them. [2]

Standard pointer.

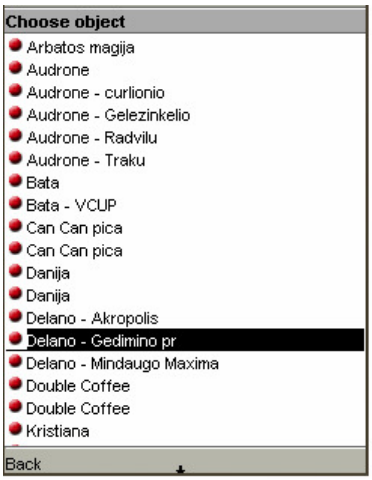
Pointer when over a point of interest.

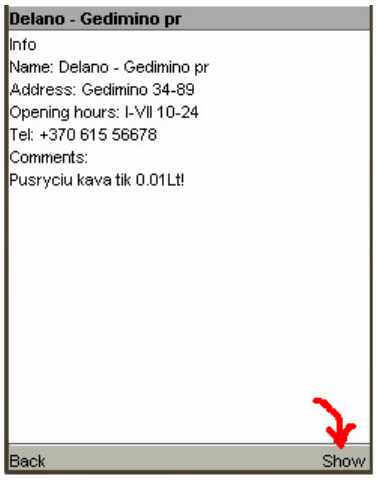
Directional pointers indicating that the map scrolls.

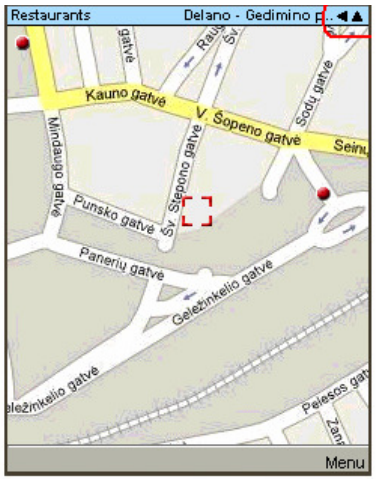
Pointer displayed when reaching the boundaries of the map.

Design discussion and usability testing regarding how to display the name of the object eventually led to the final implementation. We felt we had two different choices in this design issue, to display it in the top panel or display it near a POI. Since this information couldn't fit in the top panel, and we wanted to show the user that a POI is clickable, we decided to show the info near the POI. Our initial design involved showing the points of interest as small icons depicturing the type of the object. This design was replaced by showing all objects as a red point by several reasons. The icons would have to be of a bigger size in order for the user to clearly see what it depicts. One icon that is placed in near proximity of another icon would block the other icon out. We also found it difficult, for some of the types, to design an icon that conveys the metaphor to the user with enough clarity.

When listing all the points of interest we also wanted to give the user the opportunity to find the location on the map. This functionality was realized by showing the user in what direction to move in order to find the desired object. We felt that directly jumping to the location possibly would confuse the user and impair the user's sense of orientation (like in the picture below).





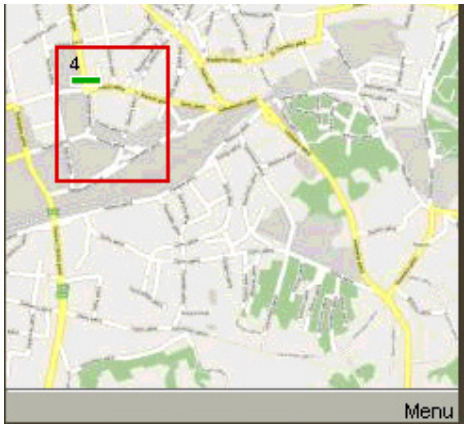


Standard point of interest.

Point of interest when under the pointer.

Arrows showing direction to the user selected destination POI.

One obvious difference between the overview- and the detailed map is the feedback about how many POIs that resides under the pointer. This is done by displaying the information as a number and a bar. The pointer also becomes more red when the number of points it is over increases.



The navigation structure for the mobile application needs to be kept as simple as possible. It consists of a simple tree structure where all the screens are accessible through one, or at most two key presses (Figure 10). A menu is often used as a tree structure and in our case the menu consists of no more than seven commands to restrict the human mind with too much information. According to Jones and Marsden (2006) it's easy to use menus because they are recognized easy in memory, but Shneiderman and Plaisant (2005) recommends that the designers should constrain the menu with less than nine commands to not overload the human mind. We have made a conscious effort to place the menu items in order of importance.

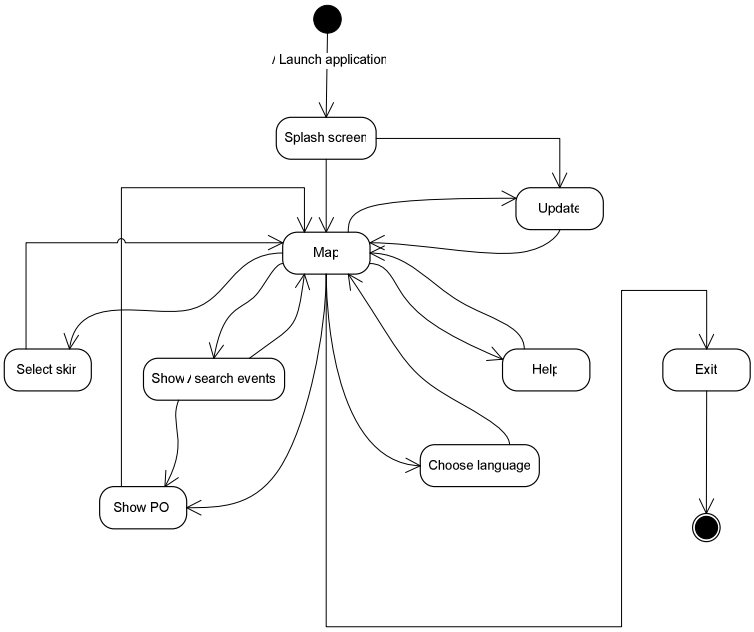


Figure 10. Simplified state diagram of the final design

Fonts

The current release of J2ME gives limited opportunities regarding the control of which fonts to use. Every phone has its own font, so we can't really control what font we want to use. We can, however, control the size of the font (small, medium, large). We initially used the medium font throughout the whole interface. However, on small screens the top panel occupied too much of the screen space. We therefore chose to use the smallest font to display the text in the panel. Despite using the smallest font some phones are still not able to display the complete name of the selected skin together with the selected destination. We therefore implemented a function to trim down the length of the text when exceeding half of the screen width. The choice of font colours is a result of an effort to achieve the highest possible contrast between the map, the panels and the text. The best way to represent text is by having a contrast between the text and the background.

5.9 CONCLUSIONS AND FUTURE IDEAS

Software development is an iterative process. That is why it is important to conduct early diagrams and perform further development on their basis. During the development of the *Mobile Shopper Guide* an early class and state diagrams were introduced. Basic classes and their responsibilities were defined which was later refined to total of 8 packages and 30 classes. On the basis of the early state diagram that defined the relationships and interaction of the classes, the functionality was build.

Deeper in the development process, some technical implementation decisions were reviewed, like connection to the database. This required a special knowledge on how to establish connection to the database from a MIDlet running on the mobile phone. It was not possible to connect to the database directly, since J2ME has no API to deal directly with Relational Databases neither locally nor remotely. It had to be established through a servlet that was put on the Tomcat Server.

Also it was important before the *Mobile Shopper Guide* was completely finished to perform an early testing, *cognitive walkthrough*, run by the project members, to detect the errors and design flaws still possible to correct for a better usability before the application gets to the end user. A further optimization was performed. A map raster format comparison was done to make sure the best format, in terms of quality and size, was chosen. Persistent data storage was tested on how many of the POIs could a single mobile phone store. This resulted in maximum 210 POIs with lots of information, like a restaurant menu. This would not happen in real life, this was an extreme assumption, but in case a city is very big we might have to look for some other information storage solutions. Final validation included testing on four different kinds of mobiles phones, either a SonyEricsson or a Nokia mobile.

During the cognitive walkthrough and acceptance test done by the end-users some ideas came along the *future* if the project was to be refined and developed further. A GPS (global positioning system) receiver would make the application better for orientation, by showing the current position and perhaps then the directions to navigate to a shop of interest. Another thing nice-to-have is a search function, so you could search for a specific area or street. Some other ideas include: landmarks on the map, showing rough areas, such as the centre of the city; a small overview in the corner of the detailed map for better orientation. The idea of the small overview map would probably be limited to PDAs rather then common mobile phones due to the limited screen size of the latter ones. Pointer could possible animate on the screen to catch attention better. Also, possibility to change the language is not implemented yet, but an “empty” menu item is currently left for further implementation.

6. CONCLUSIONS AND RECOMMENDATIONS

The thesis was based on the *Mobile Shopper Guide* application development according various analysis. The framework of the work was:

1. Business analysis to find a proper market place, justify the choice, raise the *Mobile Shopper Guide* idea and set up the differentiation strategy.

The field of specific information access was found to be the most desirable and not yet overpopulated. The *Mobile Shopper Guide* idea was found not to be completely new, however similar map-based applications specialize mostly on roads, routs, and not solely on shops. No applications exist in the market specifically for the "shopaholics". A differentiation strategy was set up and attractive features were determined: easiness to use, fast and intuitive map navigation, only basic information stored, constant air-time not required and customization by shop types.

2. Requirements' gathering stage, involving system architecture, optimization analysis for class structure, programming style and persistent data storage.

Requirements' gathering stage included a sketch of initial GUI, use cases determination which led to functionality and other kinds of requirements for the mobile platform. Further in this stage optimization analysis for the class structure, map image and persistent data storage was done: thread-based architecture and singleton pattern were admitted. Moreover, raster graphics was preferred to vector. Even though vector image would take less space than raster, it would be too time consuming and perhaps the increase in code size would not compensate the decrease in image size. Too formats, JPG and PNG were found to be the most proper, however the suitability depends on the picture itself. Persistent data storage mechanism would allow storing the data fetched from the database and therefore eliminating the need for constant air-time. The mechanism is well supported by J2ME package, thus java technology, among other features, was though to be the most suitable.

3. Selection of right programming tools by tools comparison;

Apart from java, other technologies were reviewed and compared by the most significant features, like emulation available, tool cost, platforms support, runtime speed, etc. J2ME package was selected primarily because of: free tool cost, portability throughout different platforms, graphical interface, free emulation possibilities, persistent data storage mechanism, background knowledge of java already acquired.

4. System development stage according the requirements and testing for further optimization and results comparison with theoretical expectations.

The whole system, and the mobile platform in specific, was started to develop according the state and class diagrams. Singleton single-thread architecture was optimized further to use a static

instance, rather than dynamic for the optimal instance destruction. Image name storage was optimized to a simple function, returning an image name of format 1XXX1YYY.jpg for time and space savings.

Cognitive walkthrough was made by the team members to pin-point the performance and usability flaws before the application gets to the final user. The results revealed some flaws in the design, like "update" did not clearly state what the user was updating and that the update was successful, also several navigation issues were discovered. The build validation tests showed that the RMS size was differing on various mobile phones models (e.g. on Nokia it was only 63 Kb, meaning in worst case only 120 POIs could be stored). The interactive procedure showed that different mobile phone resolutions and different capacities for the DPI (dot per inches) resulted in different map display areas. Another thing was that we had to resize the map and change the colour depth for Nokia mobile phone.

Acceptance test was run with the end-users outside the project. The results showed that generally the users were quite happy with the performance and functionality of the *Mobile Shopper Guide*, however, they found it unfortunate that the application and the menu looked so different on different mobile phones which we could not really control (we would have to make different versions for each of the model to have it exactly the same), also they missed the GPS function. They also thought of having a search function to look for a specific area or street.

The ideas offered might be considered in the future, in further development stage, but for now, since the application was aimed at basic functionality only for the best performance, too much customization and fancy features in the first version of the application would have hidden the "fast, convenient and basic" idea of the *Mobile Shopper Guide*.

LIST OF REFERENCES

[1] E. Charževskytė, D. Olekaitė. Specificity of Creating User-friendly system: mobile and web environments. // 11th Lithuanian Junior Scientists conference "Science – the future of Lithuania" material, April 9th, 2008, Vilnius: VGTU, 2008, p. 10.

[2] Gong, J. & Tarasewich. *Guidelines for Handheld Mobile Devices Interface Design. Proceedings of the DSI 2004*. Boston, USA. 2004.

[3] Sommerville, I. *Software Engineering.* Reading, Massachusetts: Addison-Wesley. 2001.

[4] Kruchten, P. *The rational unified process – An introduction*. Addison Wesley, London. 1999.

[5] Shneiderman, B., & Plaisant, C. *Designing the user interface: Strategies for effective human-computer interaction*. 4th ed. USA: Pearson education. 2005.

[6] Vartan Piroumian. *Java J2ME Platform Porgramming*. The Sun MicroSystmes Press: Java Series. 2002, p. 293.

[7] Dix, A., Finlay, J., Abowd, G.D., & Beale, R. *Human-Computer Interaction.* Pearson, Essex. 2004.

[8] Kjeldskov, J., Graham, C., Pedell, S., Vetere, F., Howard, S., Balbo, S., & Davies, J. Evaluating the Usability of a Mobile Guide: the Influence of Location, Participants and Resources. *Behavior & information Technology*. Vol. 24. 2005, p. 51-65.

[9] Mobile Internet Trends. *What is the Mobile Internet?* 2008. [Accessed 2008-05-23]. Interactive link: http://www.mobilenettrends.com/whatis/

[10] Rytų Europos studijų centras. *Apibus tvoros: Gyvenimo Lietuvoje ir Baltarusijoje palyginimas*. 2007, p. 31. [Accessed 2008-05-05] Interactive link: <http://www.eesc.lt/lt/veikla/leidiniai>

[11] World Intellectual Property Organization. *Medical Software Download to Mobile Phone.* 2006. [Accessed 2008-05-05] Interactive link: < http://www.wipo.int/pctdb/en/>

[12] Cristian Streng. *Mobile Gmaps*. 2008. [Accessed 2008-05-05] Interactive link: < http://www.mgmaps.com/>

[13] Ben Eastaugh and Chris Sternal-Johnson. *Financial Transaction on Your Phone*. 2007. [Accessed 2008-05-05] Interactive link: <http://telecompk.wordpress.com/2007/04/18/financial-transaction-on-your-phone/>

[14] Wikipedia Online Dictionary. *Mobile Development*. 2008. [Accessed 2008-05-06] Interactive link: <http://en.wikipedia.org/wiki/Mobile_development>

[15] Wikipedia Online Dictionary. *Singleton Pattern*. 2008. [Accessed 2008-05-05] Interactive link: <http://en.wikipedia.org/wiki/Singleton_pattern>

[16] Wikipedia Online Dictionary. *Image File Formats*. 2008. [Accessed 2008-05-05] Interactive link: <http://en.wikipedia.org/wiki/Graphics_file_format>

[17] Wikipedia Online Dictionary. *Binary Runtime Environment for Wireless*. 2008. [Accessed 2008-05-05] Interactive link: <http://en.wikipedia.org/wiki/BREW>

[18] Wikipedia Online Dictionary. *Mobile Information Device Profile*. 2008. [Accessed 2008-05-05] Interactive link: <http://en.wikipedia.org/wiki/Mobile_Information_Device_Profile>

[19] TonicDesigns. *SMS Chat System.* 2007. [Accessed 2008-05-10] Interactive link: <http://www.tonicdesigns.co.uk/html/sms_chat.html>

[20] IBM Research. *Mobile Computing Applications.* 2008. [Accessed 2008-05-11] Interacitve link: <http://domino.research.ibm.com/comm/research.nsf/pages/r.mobile.apps.html>

[21] IBM Zurich Research Laboratory. *Mobile Computing*. 2007. [Accessed 2008-03-06] Interactive link: < http://www.zurich.ibm.com/csc/mobile/>

[22] Maija-Kerttu Sarvas. *Mobile Solutions in Business Process*. Helsinki University of Technology, Finland. 2007, p. 12. [Accessed 2008-05-10]. Interactive link: <http://www.tml.tkk.fi/Opinnot/T-109.551/2005/reports/Mobile_Solutions.pdf>

[23] Damith C. Rajapakse. *Fragmentation of Mobile Applicatins.* National University of Singapore, School of Computing. 2008. [Accessed 2008-06-01]. Interactive link: <http://www.comp.nus.edu.sg/~damithch/df/device-fragmentation.htm>

[24] *C*. Enrique Ortiz. *Mobile Service Architecture Specification*. 2006. [Accessed 2008-04-17]. Interactive link: <http://developers.sun.com/mobility/midp/articles/msaintro/>

[25] James Matson. *Cooperative Feasibility Study Guide.* United States Department of Agriculture. 2008. [Accessed 2008-03-10]. Interactive link: <http://www.rurdev.usda.gov/rbs/pub/sr58.pdf>

[26] Danny Kalev. *Implementing the Singleton Design Pattern*. 2007. [Accessed 2008-05-01]. Interactive link: <http://www.inquiry.com/techtips/cpp_pro/10min/10min0200.asp>

[27] Liz Fulghum. Vector vs. *Bitmap Graphics – an Introductory Gguide for Clients and Designers*. 2007. [Accessed 2008-05-02]. Interactive link: <http://www.eastbywest.com/pub/vectorbitmap/>

[28] Enrico Rukzio, Michael Rohs, Daniel Wagner et al. *Application Development for Mobile Devices*. 2007. [Accessed 2008-05-17]. Interactive link: <http://www.medien.ifi.lmu.de/diamd05/slides/symbian.pdf>

[29] A Sun Developer Network Site. *The Java ME Device Table*. 2008. [Accessed 2008-05-05]. Interactive link: <http://developers.sun.com/mobility/device/pub/device/list.do?sort=manufacturer&page=1>

[30] A Sun Developer Network Site: Documentation. *What's new in MIDP 2.0*. 2008. [Accessed 2008-05-05]. Interactive link: <http://java.sun.com/products/midp/whatsnew.html>

[31] Linglom's Blog. *Accessing SQL Server on NetBeans using JDBC*. 2007. [Accessed 2008-04-28]. Interactive link: h<ttp://www.linglom.com/>

[32] Elaine Potter. *Mobile Data Implementations to Slow in 2008. Wireless Business Plans too Aggressive*. 2008. [Accessed 2008-06-01]. Interactive link: <http://www.ericsson.com/mobilityworld/sub/articles/>

[33] Sonera MediaLab. *Mobile Java Application Development*. USA, 2002, p.9.

APPENDIX A: COMPETITORS

There already exist mobile maps and maps on the Internet that you could use, for example Google Maps, Yahoo Maps etc.

Ordinary Map:
Pros
- It's free to use (if it's just the map)
- Usually you will get the whole area of a town/place to look at.
- It's not that hard to get a map.
- The most important and basic symbols are often displayed.

Cons
- It's not possible to update the map without getting a new one.

Yellow pages sites
Pros
- Easy to get information about the whereabouts of different shops and stores.
- It's possible to get a road description for different locations

Cons
- Not always accessible to the user; at least when the user is on the move.

Electronic maps
-Google map:
- Available in Europe countries: France, Italy, Germany and Spain
- Additional functions: Local business locations and contact information appear all in one place, integrated on your map
- Technology used: Ajax (Javascript and XML); map is stored in Internet
- Technical specification: supports Java-enabled phones, PC on Internet
- Free web map server application

Constant air-time, link in the mobile phone

-Yahoo map:
- Available countries: United States and Canada
- Additional functions: street maps and driving directions; Point of Interest Finder - used to find businesses and other points of interest
- Technology used: Ajax, Flash; map is stored in Internet
- Technical specification: PC on Internet
- Free online mapping portal

Constant air-time, specializes for e.g. driving directions,

-MapQuest mobile:
- Available in Europe countries: Belgium, Denmark, France, Germany, Italy, Luxembourg, Spain, Sweden, Switzerland, United Kingdom, Netherlands
- Additional functions: Instantly find your way with interactive colour maps, step-by-step instructions, reverse directions and recall of recent addresses
- Technology used: Mobile Optimized Navigation Data, ("MOND")

- Technical specification: supports GPS enabled phones
- Monthly subscription plan to MapQuest Mobile $3.99

-MapQuest navigator:

- Available in Europe countries: Belgium, Denmark, France, Germany, Italy, Luxembourg, Spain, Sweden, Switzerland, United Kingdom, Netherlands.
- Additional functions: Find restaurants, hotels, or theatres from the MapQuest.com database, Locate addresses, intersections, Customize routes by finding the fastest or shortest route, or avoid toll roads and highways, send map and directions to your phone from the MapQuest.com website
- Technology used: Mobile Optimized Navigation Data, ("MOND")
- Technical specification: supports GPS enabled phones, Web-enabled mobile phones and PDA.
- Monthly subscription plan to MapQuest Mobile $9.99
- Constant air-time required

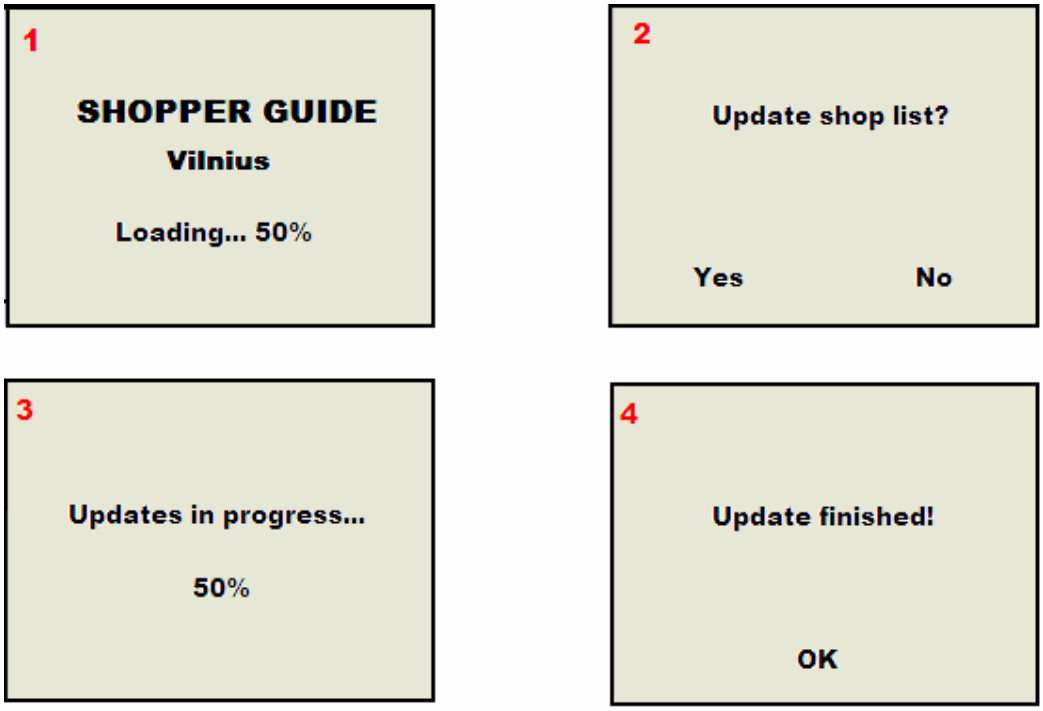
-ABmaps:

- Available in Europe countries: Austria, Belarus, Belgium, Denmark, Finland, France, Ireland, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, United Kingdom
- Additional functions: address finding, route planner, Weather forecast, Traffic information in real-time
- Technology used: streaming vector graphics and Macromedia Flash; maps data doesn't have to be stored on the device itself but downloaded from the Internet using GPRS
- Technical specification: supports Java-enabled phones
- Free Web Map Server and portal
- Quick zooming and panning, address finding
- Intuitive, smooth and friendly user interface
- Low memory usage

- Map 24

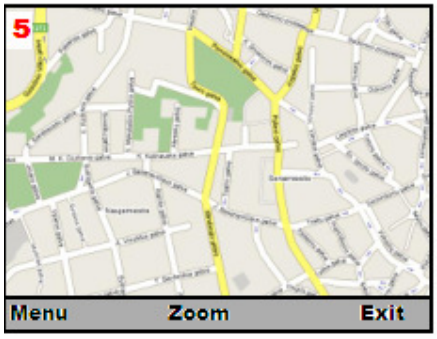
- Available in Europe countries: Andorra, Austria, Belgium, Denmark, Finland, France, Germany, Great Britain, Northern Ireland, Hungary, Iceland, Ireland, Italy, Liechtenstein, Luxembourg, Monaco, The Netherlands, Norway, Poland, Portugal, San Marino, Slovakia, Spain, Sweden, Switzerland, Turkey
- Additional functions: providing street level maps and driving directions, address and point-of-interest, interactive map
- Technology used: AJAX API combined with the Map24 core technology
- Technical specification: computer must have an up-to-date version of Java VM
- A free online mapping service
- More than 15 languages supported and easy to use
- Clear city maps

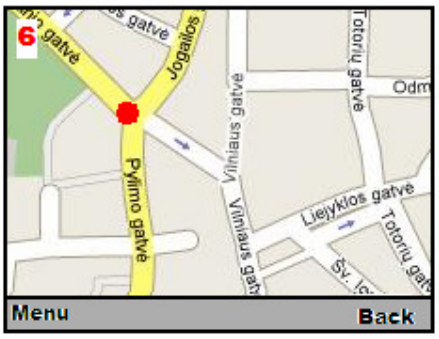
APPENDIX B: THE EARLY GUI OF THE APPLICATION

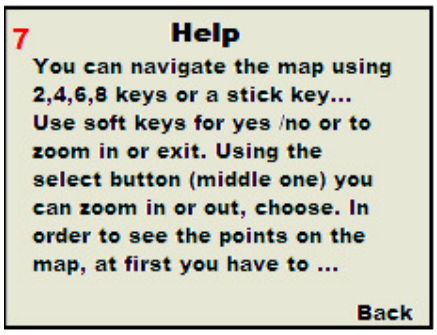


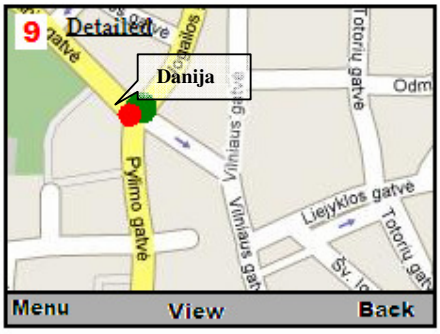
1. When the user first starts the application he/she sees the loading screen of the application.
2. When it loads, the user is asked to update the shop list. Once accepted, loading screen is shown, if discarded the map with the main menu is shown (screen 5).
3. When the connection with the database is established, the application first checks if the user has the latest updates, if they have, the updates will take much quicker. While the updates are being downloaded, progress indicating how much has already been downloaded is shown. After download is finished, screen 4 is displayed.
4. User has to press 'OK' when the update is finished. This will help to understand that the download is finished and the application will proceed to the main map (screen 5).
5. Screen 5 is the overview map with the main menu. The main menu will be a pop-up menu or a menu filling the full screen (it depends on the mobile phone used, we can't really control it). We will have the main functions in the menu; this is simplified with the 'Update', 'Choose shop type', 'View all', 'Help', The user will access this with the 'left' or the 'right' softkey. The 'Exit' button will, of course, exit the application. By using the 'joystick' or the '2','4','6','8' keys the user can navigate the box in the overview map. The 'select' button can be used to choose 'zoom' and the user will then go the detailed map.
6. In the detailed screen the user can choose to see different shop types. In the detailed map user can navigate to a certain object and view its info. The red marker is the navigation marker. User can navigate to any object on the map to see its info.

7. The help screen describes a brief user manual. It displays information about how the user controls the application. The user can access the help screen in the pop-up menu.
8. This screen appears when the user has pressed on “choose shop type” in the menu. This menu has many different shop types that the user can choose, but only one at a time, and this is done by pressing the ‘left softkey’ to accept the choice. When this is done the user will see the map with the only appropriate shops displayed (screen 9).

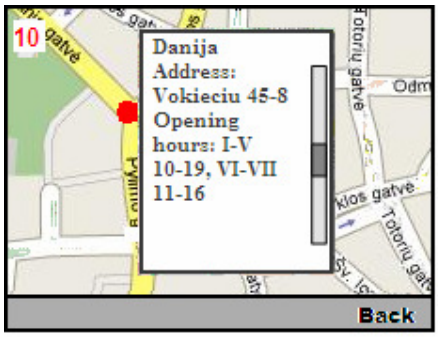


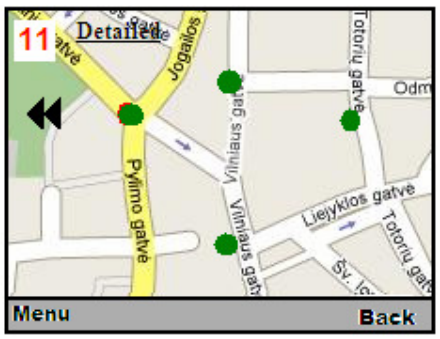


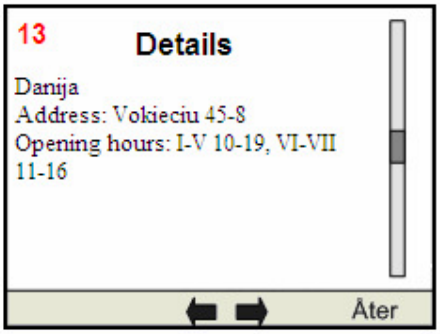




9. This screen explains what’s happening when the red marker is over a certain object. The user will see the tool-tip with the name of a certain object and the icon will perhaps change a little bit in the design to indicate it is selected. In this position the user could choose “Show” with the, in this case, the “select” button. When this happens an information screen about the object will be shown (Screen 10)







10. In this screen the user sees information about the different objects that are selected with the red marker, in this case a shoe store. The information displayed is name, address, phone number and comments. If the length of the message is too long for the mobile screen, the user can scroll with the “up” key or the “down” key. When the user has read all the information they can go back by pressing the right softkey.

11. The user can also move to the next area of the detailed map with the red marker. When the red marker is close to the corner of the screen it will change its symbol to an arrow according to the side they are trying to scroll to.

12. [optional screen] If the user presses the ‘select’ button in the main menu on the “All event” command they will be able to see all the information about the shop types.

13. [optional screen] This is almost the same screen as screen 10, but here the user can choose to scroll to the left or the right and show information about a certain object and then they should be able to see on the detailed map where the object resides.

APPENDIX C: COGNITIVE WALKTHROUGH TEST

The action sequence needed for this task is specified by the user's action (UA1, UA2, etc.) and the system's display or response (SD 1, SD 2, etc). The system display is written to see the response of the system. Two evaluators participated in this test.

The question: Download and update the skins/modules and show information about a grocery store that resides in the centre of Vilnius and then show the other grocery stores in the area left to the centre.

UA 1: Press the 'select' button, or the 'left' softkey.

SD 1: Display moves to the download page, a loading text appears and when the download is finished it moves to the 'start' menu.

UA 2: Press the 'select' button, or press the 'left' softkey.

SD 2: The overview map is shown and a red pointer is visible on the screen

UA 3: Press the 2,4,6,8 digits (or the joystick) and navigate to the centre of Vilnius and press the 'select' button or the 'left' softkey.

SD 3: The red pointer moves to the right position and then shows the inzoomed map.

UA 4: Press the select button or the 'left' softkey button.

SD 4: The 'show shop type' menu is showed on a new screen

UA 5: Press the 'select' button or the 'left' softkey button. Then press the right softkey.

SD 5: The display shows the inzoomed map with the shoe stores on the map

UA 6: Press the 2,4,6,8 digits (or the joystick) and navigate to the nearest POI and press the 'select' button or the 'left' softkey.

SD 6: The 'mouse pointer' moves to the right position and then shows the information on the POI on a new screen.

UA 7: Press the 'right' softkey.

SD 7: The inzoomed map is displayed again.

UA 8: Press the 0 digit and then press the 4 digit (or left on the joystick)

SD 8: The mouse pointer is deactivated and the screen scrolls to the next area.

The four questions asked:

1. Is the effect of the action the same as the user's goal at that point?
2. Will users see that the action is available?
3. Once users have found the correct action will they know it is the one they need?
4. After the action is taken, will users understand the feedback they get?

The evaluators answered these questions with every action completed and told a story about the usability of the system, may it be a success story or a failure story. The failure stories and some other guidelines we got from the test were then put into a failure document.

APPENDIX D: THE TEST SUBJECTS

Evaluator 1: Domantė Olekaitė **Date: 2008/05/10 14:00**

UA 1: Press the 'select' button, or the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Yes, the user would probably press the 'select' or the 'left' softkey

Question 2. Will users see that the action is available?

Yes it is definitely visible.

Question 3. Once users have found the correct action will they know it is the one they need?

Yes, I think it's obvious that they will update the application, at least the second time. There could be a problem for first time users.

Question 4. After the action is taken, will users understand the feedback they get?

Yes they will understand the feedback they get.

UA 2: Press the 'select' button, or press the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Yes, the user should want to see the map at this stage

Question 2. Will users see that the action is available?

They can see the button and see 'show map'.

Question 3. Once users have found the correct action will they know it is the one they need?

Yes it's quite clear that it's the one the user will need in order to show the map.

Question 4. After the action is taken, will users understand the feedback they get?

Yes, the overview map is shown and the user expected to see a map so this is also quite clear.

UA 3: Press the 2,4,6,8 digits (or the joystick) and navigate to the centre of Vilnius and press the 'select' button or the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

The user would definitely try to navigate with the joystick to correct area.

Question 2. Will users see that the action is available?

Yes they will see that zoom is available

Question 3. Once users have found the correct action will they know it is the one they need?

Could be difficult to find the correct way to the "centre of Vilnius", If we show some rough areas on the map, where the, for example, centre resides, it would be easier for the user.

Question 4. After the action is taken, will users understand the feedback they get?

The user will not have a problem with the feedback that they get.

UA 4: Press the select button or the 'left' softkey button.

Question 1. Is the effect of the action the same as the user's goal at that point?

It should definitely be the user's first thought. I would want to change the skins first

Question 2. Will users see that the action is available?

Yes it is visible for the user, it's not hidden.

Question 3. Once users have found the correct action will they know it is the one they need?

Yes, I would press the skins option because it's the only one that's visible.

Question 4. After the action is taken, will users understand the feedback they get?

The user shouldn't be surprised of what is visible on the screen.

UA 5: Press the 'select' button or the 'left' softkey button. Then press the right softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Confusion for the user, they'll have to mark them in order to show the skins. (F)

Question 2. Will users see that the action is available?

Yes, the buttons are not hidden.

Question 3. Once users have found the correct action will they know it is the one they need?

Maybe not, they should have some other text instead of 'show'. Why not use 'ok' instead?

Question 4. After the action is taken, will users understand the feedback they get?

Yes, if the screen shows that the skin has changed (by using the text in the left corner).

UA 6: Press the 2,4,6,8 digits (or the joystick) and navigate to the nearest POI and press the 'select' button or the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Yes I would try to navigate the marker.

Question 2. Will users see that the action is available?

Yes they will see that it's available

Question 3. Once users have found the correct action will they know it is the one they need?

If they move it it's ok, but they need information that the icon is clickable and it could be confusing that 'skin' change to 'show'.

Question 4. After the action is taken, will users understand the feedback they get?

Yes if the menu is shown and a transparent view of the map is shown.

UA 7: Press the 'right' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Yes this should definitely be the user's goal at this point.

Question 2. Will users see that the action is available?

Yes, it's not a lot of other options available.

Question 3. Once users have found the correct action will they know it is the one they need?

They will understand that they can get back

Question 4. After the action is taken, will users understand the feedback they get?

It won't be a problem, it could be better if it's transparent though.

UA 8: Press the 0 digit and then press the 4 digit (or left on the joystick)

Question 1. Is the effect of the action the same as the user's goal at that point?

No probably not, I would try with the joystick first to navigate to the left area. Or press back and then navigate to the left area.

Question 2. Will users see that the action is available?

The button is available but they won't see the hidden pointer.

Question 3. Once users have found the correct action will they know it is the one they need?

No point to deactivate first and then scroll, the users wouldn't understand that. They could also want to use back instead and navigate on the overview map.

Question 4. After the action is taken, will users understand the feedback they get?

Only if the user will see that other grocery icons are available.

Evaluator 2: Egidija Charževskytė **Date: 2008/05/10 15:00**

UA 1: Press the 'select' button, or the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Yes, I think it's logical for the user. They would definitely choose yes here.

Question 2. Will users see that the action is available?

Yes they will see it.

Question 3. Once users have found the correct action will they know it is the one they need?

Maybe the user doesn't know that you have to update the application in order to show the skin. Better information on what the update does.

Question 4. After the action is taken, will users understand the feedback they get?

It would be clearer if it says download 100%. Somewhere the user should perhaps accept and then move to the next screen.

UA 2: Press the 'select' button, or press the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Yes they would want to show the map.

Question 2. Will users see that the action is available?

Yes it's visible for the user.

Question 3. Once users have found the correct action will they know it is the one they need?

It could be easier if it said 'ok' instead of 'accept'.

Question 4. After the action is taken, will users understand the feedback they get?

Yes it wouldn't be a problem to understand the feedback that the user will get from this.

UA 3: Press the 2,4,6,8 digits (or the joystick) and navigate to the centre of Vilnius and press the 'select' button or the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Maybe, I would try to scroll to the correct area, but they would have to know where the centre of Vilnius is located.

Question 2. Will users see that the action is available?

Yes, sooner or later they will see the correct button. It's not invisible to the user.

Question 3. Once users have found the correct action will they know it is the one they need?

Yes, it shouldn't be a problem to use the joystick and push on zoom in

Question 4. After the action is taken, will users understand the feedback they get?

Yes, but they could be missing a visible scale, this will help the user understand how big of a square they have zoomed in to.

UA 4: Press the select button or the 'left' softkey button.

Question 1. Is the effect of the action the same as the user's goal at that point?

The user would probably try to navigate first instead of choosing the skins. This would not be a problem if the hotels were not showing.

Question 2. Will users see that the action is available?

Yes, they should see that it's available.

Question 3. Once users have found the correct action will they know it is the one they need?

It would be better if it said something else, 'skins' could be confusing for the user.

Question 4. After the action is taken, will users understand the feedback they get?

Yes, they would understand the feedback they get when they push the button.

UA 5: Press the 'select' button or the 'left' softkey button. Then press the right softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

They would probably want to press the 'show' button direct, so it's perhaps not the same action.

Question 2. Will users see that the action is available?

It's visible, although not us much as the user would like to.

Question 3. Once users have found the correct action will they know it is the one they need?

Only if 'show' is changed to something else and the user don't have to press both 'accept' and 'show' to change skins.

Question 4. After the action is taken, will users understand the feedback they get?

They will know if the text changes, but it would also be great if the colour of the icons will change.

UA 6: Press the 2,4,6,8 digits (or the joystick) and navigate to the nearest POI and press the 'select' button or the 'left' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

Could be a problem to navigate to the right point of interest, show more information that you can push on an icon.

Question 2. Will users see that the action is available?

Yes it's visible (although it has to be highlighted).

Question 3. Once users have found the correct action will they know it is the one they need?

The 'show' button could be confusing for the user, it would be better if the user would see 'ok' instead.

Question 4. After the action is taken, will users understand the feedback they get?

The user should see and understand the feedback that they get.

UA 7: Press the 'right' softkey.

Question 1. Is the effect of the action the same as the user's goal at that point?

If the user would want to go back, which he/she would want to, it's quite obvious.

Question 2. Will users see that the action is available?

Yes the user can see the button.

Question 3. Once users have found the correct action will they know it is the one they need?

It's the only choice the user has, so yes, they will know that it's the correct action.

Question 4. After the action is taken, will users understand the feedback they get?

Yes, the feedback that the user will get is quite clear.

UA 8: Press the 0 digit and then press the 4 digit (or left on the joystick)

Question 1. Is the effect of the action the same as the user's goal at that point?

No, the user would probably want to use 'back' instead of navigating to the left.

Question 2. Will users see that the action is available?

No, they wouldn't see the pointer turned off.

Question 3. Once users have found the correct action will they know it is the one they need?

No, it's not logical for the user. You can't expect the user to first press 'zero' and then navigate to the left. The user should be able to navigate to the left first.

Question 4. After the action is taken, will users understand the feedback they get?

The user would perhaps want some feedback on the action performed (not just changing the map).

APPENDIX E: THE FAILURE DOCUMENT

User action 1: Press the 'select' button, or the 'left' softkey.

- The text should have said something else. The user won't know what the update is for, could be for the POIs or for the application itself.
- Missing what is downloaded and if the update is successful.
- Maybe the user doesn't know that you have to update the application in order to display the shops. Better information on what the update does.
- It would be clearer if it says download 100%. Somewhere the user should perhaps accept and then move to the next screen.

User action 3: Press the 2,4,6,8 digits (or the joystick) and navigate to the centre of Vilnius and press the 'select' button or the 'left' softkey.

- It would be better if the centre of Vilnius is already visible. It could be difficult for the users to find where that is on the map. It has to be some kind of information on that somewhere.
- Perhaps, but they need to know where the centre of Vilnius is resided.
- Could be difficult for the user to know where they are. It would perhaps be easier for the user if it's connected to the big map by showing, for example, "A5" on both the overview map and the detailed map.
- Could be difficult to find the correct way to the "centre of Vilnius", If we show some rough areas on the map, where the, for example, centre resides, it would be easier for the user.
- Maybe, I would try to scroll to the correct area, but they would have to know where the centre of Vilnius is located.
- They could be missing a visible scale, this will help the user understand how big of a square they have zoomed in to.

UA 4: Press the select button or the 'left' softkey button.

- The user would probably try to navigate first instead of choosing the shop types. This would not be a problem if the hotels were not showing.
- It would be better if it said something else, 'shop types' could be confusing for the user.
- They would perhaps not see that they change hotels to groceries by pressing 'shop types'. Will they know what 'shop types' mean, it could mean something else.

UA 5: Press the 'select' button or the 'left' softkey button. Then press the right softkey.

- They would probably want to press the ‘show’ button direct, so it’s perhaps not the same action.
- Only if ‘show’ is changed to something else and the user don’t have to press both ‘accept’ and ‘show’ to change shop types.
- Confusion for the user, they’ll have to mark them in order to show the shop types.
- Maybe not, they should have some other text instead of ‘show’. Why not use ‘ok’ instead?

UA 6: Press the 2,4,6,8 digits (or the joystick) and navigate to the nearest POI and press the ‘select’ button or the ‘left’ softkey.

- They need to show that a POI is clickable. It needs to change colours and the event name has to be shown.
- Could be a problem to navigate to the right point of interest, show more information that you can push on an icon.

UA 8: Press the 0 digit and then press the 4 digit (or left on the joystick)

- No, the user will probably try to scroll from the beginning without turning the pointer off.
- It’s not visible from the beginning; you have to turn it off. This is probably not user friendly accordingly to the user
- How would they know that they could disable the pointer with ‘0’ key? It should be more visible
- If they have seen that the picture has changed, they could see the change, but it would be better if the screen show where they have been.
- No, the user would probably want to use ‘back’ instead of navigating to the left.
- No, they wouldn’t see the pointer turned off.
- No probably not, I would try with the joystick first to navigate to the left area. Or press back and then navigate to the left area.
- The button is available but they won’t see the hidden pointer.
- No point to deactivate first and then scroll, the users wouldn’t understand that. They could also want to use back instead and navigate on the overview map.
- Only if the user will see that other grocery icons are available.

New design ideas

- The map will be seen first and then they could choose a menu.
- Show big landmarks on the map.
- Solve the navigation by showing a small overview map in the corner of the detailed map.
- It would perhaps be easier for the user if it's connected to the big map by showing, for example, "A5" on both the overview map and the detailed map.
- The shop types option should be showed earlier. You could change the shop types the first time you open the application
- The pointer should animate on to the screen.
- The user could scroll between the different events and show the information on them and then show where they're on the map.
- When you move to the corner the pointer changes to an arrow and it shows that the user can scroll or it scroll automatically when you move to the end of the screen.
- Show where they have been before. Showing that A5 is to the right and A3 is to the left. It depends on how much they can scroll
- If we show some rough areas on the map where the, for example, centre resides it would be easier for the user.
- It would be clearer if it says download 100%. Somewhere the user should perhaps accept and then move to the next screen.
- Language and help can be accessed everywhere.

APPENDIX F: THE ACCEPTANCE TEST

Test person: **Mobile phone:**

MODULE 1: Update application

Task: The user should connect to the database and update the application.

Form based Script	Correct	Date	Initials
1. Is the text visible enough, is it using an acceptable colour scheme and is the size ok?			
2. Is the text logical and understandable for menus and buttons?			
3. If the help text is used, is it to any use for the function?			
4. Are the warning messages understandable?			
5. Does it take less than three seconds before the application gives a response to a pressed button?			
6. Are all buttons available in order to execute the function?			
7. Does the function work in a correct way without giving failure messages or crashing the application?			

The general questions

Question 1: How was your experience with the tourist service/application?
Question 2: What positive thoughts came from using this application? Why?
Question 3: What negative thoughts came from using this application? Why?
Question 4: How did you experience the navigation of the application (to move around and find information)?
Question 5: Do you feel that the phones that you used were suitable for this service?
Question 6: Do you think that you could start using this service as a tourist or as someone who already lives in a city?