2007:107

MASTER'S THESIS

Implementing a Scratch Ticket Client for Mobile Devices

Christian Nilsson

Luleå University of Technology

Master's thesis
Computer Science
Department of Computer Science and Electrical Engineering
Division of Computer Communication

2007:107 - ISSN: 1402-1552 - ISRN: LTU-DUPP--07/107--SE

Implementing a Scratch Ticket client for Mobile Devices

Christian Nilsson

24th October 2007

Abstract

With the increasing capabilities and the fast growing ubiquity of mobile devices in the community, the mobile market has the potential to become an important target for advertising. Today Netix AB runs the web service Skraplotto.se where users are given the opportunity to scratch tickets for free and win prises online. The prises are paid for by the advertising companies in an attempt to market their products. Recognising the growing mobile market and the potential therein, Netix has expressed interest in having a mobile version of their scratch ticket concept, Skraplotto.se, developed.

This master thesis has in cooperation with Netix resulted in a ready to deploy application for Java 2 Mobile Edition (J2ME) capable mobile phones and other compatible devices, thereby allowing Netix to take their marketing concept one step further and enter into the world of mobility.

Acknowledgements

I would like to thank all the Netix employees for taking me in and making me feel like one of the gang. I really appreciate being given the opportunity to perform my master thesis in cooperation with you.

I would also like to thank Britta who has supplied me with some of the graphics for this report. Without her, some of the images in this report would either look a lot worse, or more likely, violate copyright laws.

Last but not least a huge thank you goes to Mobizoft AB who let me borrow their phones and PDAs in order to test my applications compatibility with different devices.

Contents

Chapter 1

Introduction

1.1 Background

Skraplotto.se is a site run by the company Netix AB where people can get free electronic scratch tickets and win prises from the companies advertising on the tickets.

The concept of free tickets paid for by the advertising companies has received a positive response both from the advertisers and the users of the web page. The concept offers companies the opportunity to associate their products with the excitement of scratching the tickets. It also gives the possibility to hand out sample goods or services to people, without risking the perceived loss of value of the product which can occur when samples are given out for free to the public.

Most of the Swedish population have access to a computer with an Internet connection. The consumption of scratch tickets in Sweden is also rather large. This led to the decision to develop an electronic scratch ticket accessible through web browsers, however there were also thoughts about bringing the concept to mobile phones as well.

The mobile phone industry predicts that during the next few years many more mobile services will be deployed and much of the work done on ordinary computers today will change and instead be performed on mobile devices and phones. It is already possible to buy and scratch tickets on your mobile phone, and being the first to supply free tickets to mobile phones might lead to a large advantage over the competition. The market might be even bigger than the one for ordinary computers; since the mobile phone is a device most people always carries around with them, it would make the service a lot more accessible.

1.2 Purpose

The purpose of this project is to produce a ready to use application designed for mobile devices, mainly mobile phones, which can be used in conjunction with the existing infrastructure of servers with a minimal change to the existing server software.

- Evaluate different mobile platforms in order to find the one best suited for current and prospective users of Skraplotto.se. This includes aspects such as device capabilities and pricing.
- Select the elements considered necessary from the existing web page in order to give the mobile client the desired functionality and ease of use.
- Implement a client for mobile devices with the necessary functionality to scratch tickets fetched from the existing servers.
- Attempt to maximise compatibility of the client with devices using the selected platform.

1.3 Delimitaitons

While the goal of this work is to present a finished and fully functional mobile client to Netix AB, a couple of delimitations have been defined.

- There is no way to guarantee 100% compatibility with all the targeted devices without a huge amount of testing on different hardware. There may also be differences between firmware versions on the same units, making it practically impossible and incredible expensive to test all variations. Still the application should be implemented in such a way that it has a large probability of working on untested devices.
- It will only be possible to scratch normal tickets on the cell phone. There is a special kind of ticket used in conjunction with marketing polls, however this feature will not be implemented. The reason for this is that the time of writing that functionality has not yet been implemented on the back end.

Chapter 2

Skraplotto.se infrastructure

2.1 Server platform

The Skraplotto web page is served from a Linux based server running Java Enterprise Edition which in combination with Apache Tomcat is used to generate the page content and serve it to the web clients.

All the data needed by the system, such as user names and lottery ticket information, is stored in a MySQL database. The database is also hosted on a Linux-based server.

The data stored in the database allows Skraplotto.se to among other things target an advertisement campaign to specific users based on for example age and city of residence. The system also tracks every ticket a user has scratched in order to be able to track which adverts the user has been exposed to.

2.2 Data transport

The web page is created using a technique known as AJAX which consists of a combination of Java scripts and XML. Ajax is intended to make web pages feel more like a real stand alone application than a classic web page. This is accomplished by exchanging small amounts of data with the server in the background and changing the page content without having to reload the entire page for every requested change.

All the information shown on the screen is transferred using the Hypertext Transfer Protocol (HTTP). The front end uses Java script to create the web page. Requests are sent to the server from the Java script and the server responds with XML formatted text documents. These documents contains the dynamic data which is to be shown by the Java script and also information on where to find the different images and what images to use when presenting the data.

2.3 Security

In order to access their tickets users log into the site using either the email address supplied at registration or their personal number, combined with a password. The login details are sent in plain text, making them vulnerable to interception by unauthorised third parties. However even if an account is accessed by an unauthorised party there should be no real loss to the account owner, as all the more valuable prises are shipped to the national registration address of the registered user instead of the address entered in the user details. By shipping to this address the company assures that the holder of the registered personal number also receives the prise.

Chapter 3

Client functionality

3.1 User operations

The client should provide the user with most of the possibilities available on the website. Some options will not be available for different reasons, for example there will be no possibility to edit user details because of security concerns among other things.

- Log in, retrieve and scratch the available tickets.
- View a list of won prises.
- Receive an overview of some of the prises currently available.
- View a list of people who won something recently.
- Change some basic user settings.
- Read some general information about the service and the company.
- Exit the program.

3.2 Client functions

In order to get a fully functional client the following components were considered important. Some of the functions are of course vital, but some others are also included in order to make the mobile client capable of performing some of the more important tasks from the web site.

There are also some tasks that were considered unnecessary to support in the mobile client, such as user registration and modification of personal data.

Application start

Once the application is started the main menu is shown. The user can then make a choice from this menu.

Get tickets

The application checks if the user is logged in, if not the application attempts to login the user. If the login attempt is successful or the user already was logged in the application will present the users with the available tickets to choose from.

Scratch a ticket

Once the user has selected a ticket the application retrieves the necessary data and presents it to the user. If the ticket contains no prise the application returns to the get tickets step. If the ticket is a winning ticket the user will be asked to answer a knowledge based question. If the user answers correctly, he/she receives a congratulatory message. If the answer is incorrect the user is informed that he/she unfortunately got the answer wrong and that there will be no prise.

Display available prises

The application downloads a list of available prises and displays them to the user.

Display won prises

The application checks if the user is already logged in, if not a login attempt is made. If the login attempt was successful or the user already was logged in, a list of prises earlier won by the user is fetched from the server and displayed.

View earlier winners

A short list of people who has recently won something is downloaded from the server and displayed to the user.

Modify settings

The user is allowed to modify the user name and password login details. The user should also be given the option to save the password between application restarts.

Quit

The application closes any remaining network connections and frees other resources before shutting down.

An overview of the application flow can be seen in figure 3.1 on the following page.

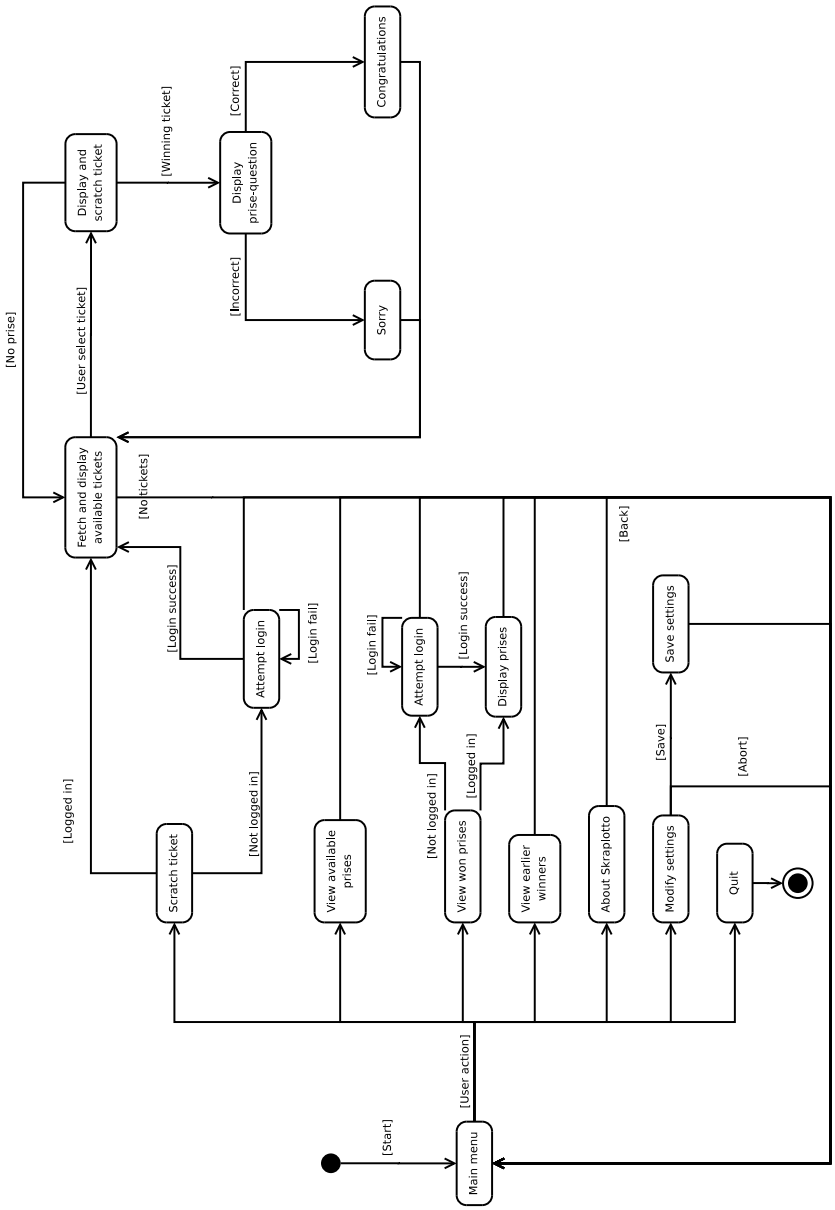

Figure 3.1: *An overview of how the mobile application is expected to function.*

Chapter 4

Choice of platform

There are several platforms to take into consideration when developing an application for mobile devices. Which to use depends on what kind of capabilities one requires from the phone, as well as the targeted user groups.

This is list of the commonly available systems for mobile devices:

- Blackberry
- Flash Lite
- Java 2 Micro Edition
- Linux
- Palm OS
- Symbian OS
- WAP
- Windows Mobile Smartphone

Out of these it was decided that Blackberry, Linux, Palm OS and Windows Mobile Smartphone were of less interest than the rest, mainly due to a considerably smaller user base. Unfortunately information regarding sale numbers of mobile devices is hard to get hold of. Because of this the user base size is based on approximations drawn from real life experience.

4.1 Selecting the platform

The remaining alternatives were selected for a closer evaluation. In order to compare the different platforms a few points were set up to decide how well the platform in question would fit the requirements.

4.1.1 Grounds for evaluation

Device and user base

In order to reach as many users as possible the platform should be supported by several manufacturers and be implemented on a large number of mobile devices.

Most of the registered users on Skraplotto.se are between the ages of 18 and 35 years old, in other words the selected platform should be common among people in this age span. Unfortunately no reliable data regarding mobile phone sales and user ages could be found to base the selection on. Instead it was estimated that support among devices up to about two years of age and in the lower price segments should cover a majority of the intended user base. Additionally by supporting the less advanced devices, more advanced units should hopefully also be able to run the application.

Functionality and Computational power

The mobile devices should have enough computational power to for example display and scale images.

Connectivity

As this is a service which requires Internet connectivity, the devices must have the possibility to connect to the Internet. The faster the connection the better.

While the transfer rates offered by GPRS should be sufficient, faster connectivity such as the capabilities offered by for example 3G/UMTS would be an obvious advantage.

Interactivity and looks

In order to appeal to the users it must be possible to make an application that looks good and feels intuitive to use. This includes input methods and also the quality and size of the screen of the mobile devices on which to run the application.

4.1.2 Flash Lite

Flash Lite is a software supplied by Adobe which during the last year (2006) has begun finding its way into more and more cell phones and other mobile devices.

The graphic capabilities are excellent and applications can be made very interactive. Connectivity depends on the underlying platform which can be Symbian, Windows Mobile and also manufacturer proprietary systems such as some Sony Ericsson phones.

A major drawback is that this is a technology which has only recently become available on mainstream phones and there are still many new phones which do not support Flash Lite. At the time of writing Adobe claims support for about 300 different devices, however many of these are phones exclusively made for the Asian market and not available in Sweden [1].

4.1.3 Java 2 Micro Edition

Java 2 Micro Edition (J2ME) is a version of the Java Virtual Machine developed by SUN Microsystems designed to fit into the tight constraints of for example mobile devices. The J2ME applications run in a virtual machine, much like the standard Java available for desktop computers.

J2ME is comprised of a set of configurations and profiles. There are two different configurations for J2ME, Connected Limited Device Configuration (CLDC) being the one implemented in cell phones. There are two versions available, CLDC 1.0 [2] and CLDC 1.1 [3]. The configuration in conjunction with a Mobile Information Device Profile (MIDP) makes up what is the Java runtime environment implemented in for example Java compatible mobile phones. There also exists two versions of the profiles, MIDP 1.0 [4] and MIDP 2.0 [5], the later being the one implemented in all Java compatible phones sold at the time of writing.

J2ME is supported by very many different mobile devices all the way down into the lowest price segments. According to Sun Microsystems currently more than 600 different models, of which more than 400 support MIDP 2.0, are capable of running J2ME applications. The large amount of compatible devices unfortunately leads to a great deal of differences in respect to hardware capabilities among the different devices.

Connectivity depends on the underlying device, however phones supporting Java are also expected to be able to communicate over the Internet. Almost all phones should at least support GPRS and some also 3G/UMTS for even faster data transfers.

4.1.4 Symbian OS

Symbian is an operating system designed for mobile devices from the start. It is foremost aimed at smart phones and is designed to be very memory efficient. It is among others used by Nokia and Sony Ericsson for their most advanced phones.

Connectivity on Symbian devices is equal or better compared to that of J2ME devices, supporting GPRS and sometimes also UMTS. Additionally some devices have WLAN support providing very good speeds and low latencies for wireless Internet access when in the vicinity of an available access point.

There are currently about 100 different phone models being sold running Symbian, all of which have excellent media capabilities and computing power, some of them even incorporate a touch-sensitive screen which could provide very good user experience. However most of these phones are more advanced phones, smart phones, which usually carries a heftier price tag [6]. Symbian phones can also run J2ME applications making it less interesting to produce a Symbian only client.

4.1.5 WAP

The Wireless Application Protocol (WAP) is basically a slimmed down version of HTML optimised for mobile phones. Looking at the number of supported devices WAP is most likely the alternative with the absolutely largest number of compatible

devices. However without java script or similar functionality on the WAP-pages the user experience would suffer from very static content.

4.2 Selected platform

After serious consideration it was decided that J2ME would be the platform on which to implement the application. More specifically targeting MIDP 2.0 compatible phones.

4.2.1 Motivation

J2ME combines a very large user base with reasonable processing and graphics capabilities. It is also available in cheaper phones, something which was considered important in order to be able to reach as many customers as possible.

The MIDP 2.0 specifications were chosen since they provide access to some features deemed necessary to implement the application. The earlier standard MIDP 1.0 does not support features such as full screen graphics nor is any form of transparent graphics guaranteed to work.

An added perk with using J2ME is that many of the more advanced platforms also implements J2ME support either native or possibly with the help of 3rd party applications. This will hopefully enable the application to run on a very large selection of mobile devices reaching the largest possible user base.

4.2.2 Foreseeable problems

The largest advantage of J2ME is also the largest drawback; the number of devices supporting it is very large. Since SUN does not provide a common Java implementation to the device manufactures, the manufacturers themselves have to implement the Java environment on their devices. This can lead to different interpretations of the specifications and thereby slightly different functionality on various devices.

Chapter 5

Design and Implementation choices

5.1 Development environment

The NetBeans IDE was chosen as development environment, mainly because NetBeans is already in use by other programmers at Netix AB but also because the author of this report already had experience in using it. NetBeans is a free Java-based open-source development environment which is available for most modern platforms.

While NetBeans in its base configuration is intended for developers of desktop Java applications, there exists several addons made to suit different environments. One of these addons, which was also used in the development of this application, is the NetBeans Mobility Pack. This addon is a well integrated system for developing applications for J2ME devices, including emulators and on device debugging.

5.2 User interface

Developing a user interface for a hand held device is quite different from implementing user interfaces for normal stationary computers. The screen size and resolution is usually a mere fraction of the size of a normal home computer screen. Mobile devices also have less convenient input methods usually limited to a numeric keypad, a four-way pad or joystick and a few extra buttons surrounding the screen.

5.2.1 Technology

There are two possible paths to follow when developing graphical user interfaces for the MIDP environment.

The first option is to use a form layout with a set of standard components such as buttons and text fields etc. When using these components the device handles the layout

and dictates the look of the components, it also handles all user input. An advantage with this approach is that the interface usually resembles the general interface of the device making users feel at home with the familiar looks. This solution frees the developer from having to worry about screen sizes and layout of the components, usually leading to faster development and easier porting between different devices.

The second option is to use the Canvas class and designing everything from scratch. This method is commonly used in games, since it allows the application to draw directly to the screen. While this allows the interface to be completely customised, the components available in the previous option can not be used on the Canvas. This means that the application is required to draw all user interface components and also repaints the screen when necessary. The application also has to handle all the user input since it is not taken care of by the device.

In the original plans it was decided that the application should hold similarities to the website such as colours and as many other graphic elements as possible in order to strengthen the trademark. The website is designed to look and feel more like an ordinary stand alone application than a web page and in the same way it was also desirable that the mobile application reminds more of a game than a static screen with information. In order to accomplish this goal the canvas approach was deemed to be the most suitable method to implement the user interface.

5.2.2 User interaction

The question of how much user interaction should be required by the application is a difficult question. From the user perspective it should be easy to scratch the tickets and require as few key presses as possible.

On the other hand the whole idea with Skraplotto is to expose the user to advertisements and that is what the advertising companies pay for. In other words the user should have to look at the screen and be active while playing.

Three levels of user required user interaction were discussed during the design stage. Out of these only the one decided on were implemented.

- *Very low interaction*
  The user basically pushes a button and the application scratches each square in series. Practically nothing is required by the user who does not even have to look at the screen. It is very easy to scratch the tickets, but it does not expose the users to the advertisements, nor does it really have any entertainment value.

- *Medium interaction*
  The user is presented with a view of the nine squares and can choose which to scratch. Holding a button down will scratch the square and releasing the button will stop. The user may choose to only "peek" at the content of a square by releasing the button early and thereby only scratching some of it. Some interaction is required and the user is required to watch the screen while playing.

- *High interaction*
  The nine squares are presented to the user, who can scratch them by moving a

"pointer" across the screen using the phone keyboard or joystick. The surface below the pointer is cleared and exposes the advertisement hidden behind it. This requires a lot of interaction from the user and also closest resembles the web page and how a real scratch ticket is played.

Out of these choices the middle route was chosen to be implemented. The reasoning behind this was that while the third option would hold the closest resemblance to a real lottery ticket, it was deemed to make the ticket scratching too much of a hassle. The first one limits the exposure of the message from the advertisers severely while it was also considered less likely to entertain the users because of the low level of interaction.

5.2.3 Scratch surface layout

The layout of the application needs to both provide a good view of the advertisement at the same time as it gives the user a good overview of the possible winnings already revealed.

Four different layouts, which can be seen in appendix C, were drawn up and discussed with the co-workers. It became clear that looking at images alone would not give a fair representation of how the different layouts would work in reality.

Therefor the four suggested layouts were implemented and run on a mobile phone in order to evaluate the different solutions. In the end the classic look with nine equal sized squares arranged in a three by three pattern was the one decided on. The demo of the layout in question can be seen in figure 5.1. The reasoning behind the choice was that the layout was familiar and looked like a scratch ticket was expected to look. Moving around among the squares selecting which to activate also felt natural and intuitive.

Figure 5.1: *The decided upon layout for the ticket, a classic three by three grid with equal sized squares.*

5.3 Session handling

Since the HTTP protocol has no built in functions to keep track of states, cookies is the usual way of handling session information for web pages. Cookies usually works by the server sending a random session identifier to the client using a cookie. The client then includes the cookies during subsequent requests to the server, allowing the server to keep track of a large number of unique visitors.

Skraplotto is designed to utilise cookies to keep track of logged in users. In order to minimise the amount of changes made to the server software, the mobile client should also support cookie handling in order to identify itself to the server. Unfortunately there are no built in support for cookies in J2ME, so some basic features to handle the cookie containing the current session needs to be implemented.

5.4 Data transfers

The client receives data from the server using XML-formatted text. It is then up to the client to parse out the relevant parts of the data in order to display it or fetch images referenced in the data. Images are retrieved from the server using ordinary http-requests for the specific file.

A huge advantage with using plain text for information exchange is that it allows for easier debugging than using binary encoding. The information can easily be read during testing without the need for a special tool to translate it into human readable format. On the other hand the overhead of sending XML formatted data can become quite large when for example dealing with small amounts of data where XML tags constitutes a large part of the payload.

A possible way of reducing overhead in the data transfers would be to implement a binary encoding method for the traffic from the server to the client. However the amount of overhead is considered too small to merit the time needed to implement some sort of binary encoding. Implementing an additional encoding method would also lead to changes having to be made to the server. In this project however it is desirable that no changes should be required on the server for the mobile client to work.

5.5 XML parsing

As described in 5.4, the client downloads information from the server as XML-formatted text. In order to display the relevant data the application has to parse out the desired parts from the XML files. Unfortunately there is no built in support for XML parsing in the J2ME version used. There does exist an XML parser in some of the more recent implementations, however a smaller subset of J2ME was chosen in order to maximise compatibility.

The two alternatives remaining in order to parse the XML files are either to implement a solution of our own or to use an existing implementation. An article on the subject by

Sun Microsystems [7] provided a number of different parser implementations available to J2ME. It was decided that none of the parsers quite contained the desired functionality. Some searching revealed an additional alternative, the Sparta XML parser [8] was found.

While Sparta has not been updated since the end of 2003 it still had a large advantage over the other parsers found - it supports a subset of the XPath language. XPath is an expression language which allows the application to access different parts of the XML structure. The XPath-support was considered an advantage since parts of the web-client also used XPath functions to process the data. Using the same system would make it easier to carry over changes from the server and web-client to the mobile client than if a different parsing method would have been used.

Chapter 6

Encountered problems

This chapter describes some of the problems encountered while developing the application. Some of the problems are inherent to the J2ME platform while some were caused by the device on which the application was tested. Additional information on the devices used for testing can be found in Appendix D.

6.1 Image formats

Today most of the bitmap graphics on the web are in GIF and JPEG format, this is also the case with the graphics at Skraplotto.se. However the specifications for Java 2 ME only requires the device to support PNG for bitmap images. While some devices also support both JPEG and GIF, the application should not assume support of anything above what the specifications demands.

In order to find out what formats are supported by the device in question one can query for supported image formats. This unfortunately proved to give incorrect responses, with some devices claiming to support formats it later could not load, and also the other way around with devices supporting more formats than the response indicated. The solution to this was to include a pair of images which the application tried to load and depending on the result the capabilities of the device could be determined.

For devices not supporting JPEG and/or GIF it was first considered to include a decoder for the two formats in the application. Freely available decoders were found and tested for both formats. Unfortunately they proved to require a lot of resources, both in memory during run time and size of the application. The run time memory requirements of the application increased by more than 100% from about 400 kilobyte to close to 1 megabyte even when decoding small JPEG images. While this amount of memory is available in several devices, it still excludes a large number of cheaper models. In addition the delays due to the decoding of images were considered excessive in some cases.

Because of the added overhead it was decided that images in PNG format should be made available by the Skraplotto server instead in order to minimise the amount

of image processing performed by the device. At the moment it is performed by a small script which downloads all the images to a different directory on the server and converts them into PNG images. Eventually the image conversions should be performed directly by the web server depending on the file name in the HTTP request as it would require no involvement from the administrator.

6.2 Image transparency

One of the major reason for choosing MIDP 2.0 over the earlier standard MIDP 1.0 was the ability to handle transparency and alpha channels in images. With this also came support to access the colour values of individual pixels in an image. These features was deemed crucial in order to get some of the features in the application to work as intended.

The main reason for the importance of raw pixel data access was the need to scale images to fit. A very basic scaling algorithm was implemented in the application and worked fine initially. However once images containing transparency data were scaled the transparency information was lost. This problem was traced down to a call to the method `drawRGB()`, an API call which transforms an array of integers describing individual colour values to horizontal lines of pixels in an image object.

To solve the problem with transparency information disappearing an other function for manipulating images had to be used. The method `createRGBImage()` also takes an array of integers describing pixel colour values, but it requires the entire image representation to be present in the array instead of being able to assemble the image one line at a time. Since the scaling algorithm works one line at a time this caused some overhead as the entire image had to be in memory twice, both as an integer array representation and as an image object.

Another problem found on some Nokia phones was that they had problems with reading pixel data from an image into a integer array. Instead of returning meaningful transparency data from the pixels, they all came out opaque. A solution to this problem was to draw the image on two different backgrounds, one white and one black. The images were then compared to each other to find the pixels where they differed. The pixels which were different were the ones that should be transparent and the alpha data in the integer array where set to the appropriate value.

At a later stage in the development it was decided to add some additional graphic features by using an additional API from Nokia called NokiaUI. This API is supported by basically all Sony Ericsson and Nokia cell phones and allows some extra eye-candy in the application by additional use of alpha channels. Since not all phones support these extra features a test is performed during the application start up to examine if the API is supported or not. If it is not, more basic graphics is used in some places.

Initial testing was done on a Sony Ericsson K700i phone which supported all the used features. Using both a Series 40 and a Series 60 emulator from Nokia also gave the expected results. However once the application was tested on a set of real Nokia phones some problems surfaced. The older and more basic Series 40 phones worked fine, while the more advanced Series 60 phones had problems creating empty transparent images

and drawing text on them. A simple solution was implemented where pixels that had not been changed from the background colour were changed to transparent after the drawing was done.

6.3 HTTP redirects

While a user is scratching a ticket on the web he/she can also click on an advertisement to open the advertisers home page and receive additional information about the prise. When an advertisement is clicked the request is first sent to the Skraplotto server tracking which advertisement was clicked. Once the click has been stored the users web browser receives a http redirect pointing it to the advertisers home page.

On mobile devices the advertisers site is displayed by the build-in web browser. During testing this proved to be a problem since not all browsers function the same way and have the same features. On some of the devices tested the internal browser did not follow the redirect and instead only displayed a blank page. This lead to the decision to handle the redirect inside the client.

A small addition was made to the server software allowing the client to receive the URL of the redirection when performing the click request. This change was motivated by the fact that while the current http redirects could be handled in J2ME, there were still the problem with a possible change to Java Script redirects on the Skraplotto pages. Thus a small addition to the server software would provide a more future proof way of handling it in the mobile client.

6.4 TCP latencies

6.4.1 Background

A problem which was not considered when starting the implementation was that of the latencies inherent to the mobile connection. Initial development and testing were performed in a simulated environment where network delays are similar to that of a normal ethernet connection. Once testing of the networking parts were moved to actual mobile devices a large increase in delays were noticed.

These delays were considered to be a big problem. Fast interaction is an important part of the application and the user should not have to spend a lot of time waiting for the data to be loaded. Initial tests showed that the delay for a single http-request could be as long as up to four seconds. Worst case scenario is that the client will have to perform five requests to the sever to download the images for a ticket, this would require at least 20 seconds in addition to the time required to transfer the data.

Existing information [9] indicated that one-way delays of close to one second would to be expected. Since HTTP uses TCP which implements a so called three way handshake (as seen in figure 6.1 on the next page), no data is sent from the server until the fourth message. Based on this, the one way delays of one second stated were quite in line with the current findings. Unfortunately the existing information did only cover traffic

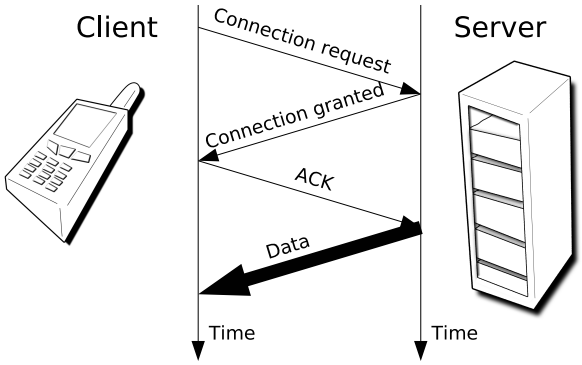

Figure 6.1: *TCP Three-way handshake*

over GRPS and not data transfers over 3rd generation mobile networks. In order to get a more up to date view of what kind of delays to expect over different types of connections further testing had to be performed. To help design these tests two possible solutions to the latency problem were conceptualised.

File concatenation

The server could join a bunch of small files into one larger archive for transmission, thereby reducing the number of http connection setups to one or two. This would however require some additional functionality to the server and it would also increase the load on the server because of the archive creation.

For archiving the Java Archive (JAR) format was deemed suitable since it is already available in the server software. It is however not available in the J2ME environment on the client, but after investigating the file format it was decided that a simple unarchiver for non-compressed JAR-files should not be too difficult to implement in the client application. The additional code in the client would increase both program size and memory requirements. Memory usage would suffer mainly because the application would be required to hold both the archived images and a representation of the extracted image in memory during the extraction. There is also a certain overhead when archiving using the JAR format, leading to more data to transfer.

Parallel transfers

The other method of decreasing the impact of TCP latencies considered was to send several http-requests to the server at the same time. While this would not reduce the number of requests it would at least cause the requests to be performed more or less simultaneously thereby hopefully reducing the total delay. Performing several requests in parallel to the server would cause a somewhat increased load on the server, but since the requests originate from a wireless device there should be some jitter introduced anyway, thereby spreading the requests somewhat over a short period of time.

To be able to perform several requests simultaneously the networking needed to be expanded from a single thread to the number of requests to be performed. Since the program was already designed to put the requests in a queue serviced by a download thread, extending the functionality to several threads should not be too difficult.

6.4.2 Simulation of possible solutions

In order to evaluate the impact of the two alternative solutions to the TCP latency problem a small test application was implemented using J2ME. It would connect to the server and download five 7500 byte files. The application did also download one 37500 byte file in order to evaluate the performance gains when using the concatenated approach. The files contained random data in order to mimic the transfer of high entropy compressed image files. The size of the test files was decided based on the average size of the images of all previous advertisement campaigns to date. The tests were performed during low traffic hours to reduce the impact of network utilisation.

Theoretical transfer times were approximated using a formula derived from equations for HTTP delays with multiple parallel connections in *Computer Networking* [10].

$$
\begin{aligned}
\textit{Response time} &= (M * O/R) + 2(M/X)RTT \\
\textit{Where} & \\
M &= \textit{Number of objects} \\
O &= \textit{Object size} \\
R &= \textit{Transfer rate} \\
X &= \textit{Number of connections} \\
RTT &= \textit{Round Trip Time}
\end{aligned}
$$

GPRS

Tests showed that ICMP Ping round trip times were quite stable around 0.7 seconds on a good GPRS connection during low traffic hours. Round trip times during office hours were usually higher and more varied. GPRS tests were performed on a Sony Ericsson K700i cell phone which ideally will be able to reach a down-link speed of 80 kbit/s.

As can be seen on figure 6.2 on the following page the transfer time can be drastically improved by running more than one transfer in parallel. With four or five parallel transfers the total time needed is cut down to about 11 seconds, less than 50% of that needed when transferring all the files sequentially.

In comparison, a simulation of using the concatenated files approach by transferring a single 37500 byte size file required on average almost 18 seconds using the same setup.

The calculations shows that much less time should be needed, but it must be taken into consideration that the transfer rates, even when using one large file, does not come close to the theoretical 80 kbit/s maximum. Various kinds of interference may also affect the device with reduced throughput as a result. However as can be seen in the more exhaustive data in Appendix A, the transfer times for GPRS are quite uniform,

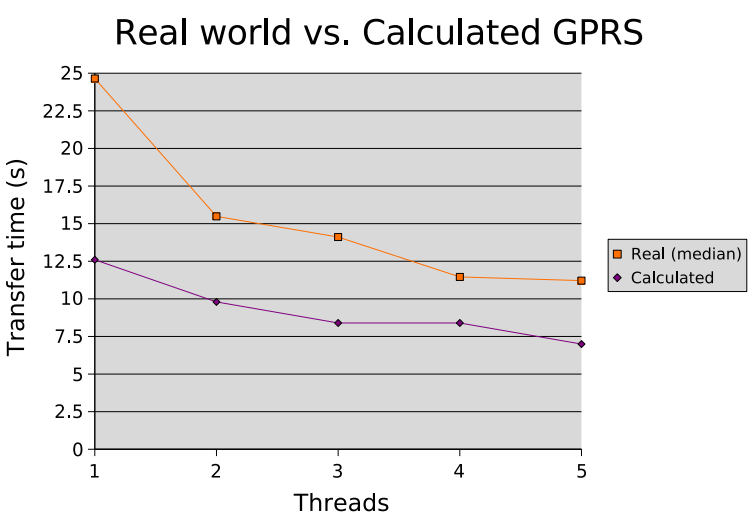

Figure 6.2: *Average GPRS transfer times for different number of parallel threads compared to calculated transfer times. Based on 20 runs of five 7500 byte files.*

indicating that they are probably not being affected by random noise. Investigating the various causes of this discrepancy is unfortunately beyond the scope of this project, but the results are similar to real world usage of GPRS based services experienced by the author.

3G/UMTS

The 3G/UMTS tests were performed using a Sony Ericsson K800i cell phone. Using the available UMTS network service this device should be able to reach a down-link speed of up to 384 kbit/s, seven times faster than the GPRS device. Round trip times were measured to about 0.25 seconds during low traffic hours. Similarly to the GPRS device the round trip times were longer and less stable during office hours.

As shown in figure 6.3 on the next page the transfer times using the more modern and faster 3G network are improved noticeably. The same trend can be seen where more threads lead to shorter transfer times. The peak when using four threads can most likely be attributed to the very varying throughput seen when performing the tests. Looking at the data in Appendix B it can clearly be seen that the performance is a lot less consistent than the results in the GPRS-test. The variations are likely due to the reduced tolerance of interference brought on by the higher data rates.

Comparing the times needed by the parallel transfer approach to the time needed to transfer a single 37500 byte size file gives quite similar results with an average transfer time of 3.5 seconds. The concatenated approach might be slightly faster in most of the test runs, but the waiting times are still well within reason even when using parallel transfers.

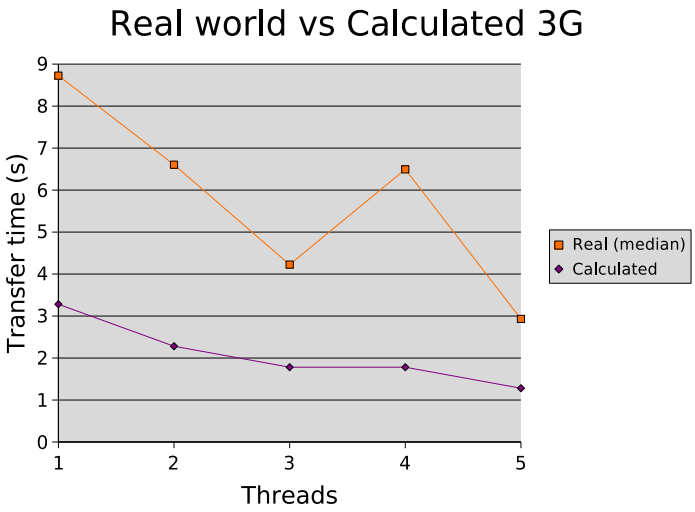

Figure 6.3: *Average 3G transfer times for different number of parallel threads compared to calculated transfer times. Based on 20 runs of five 7500 byte files.*

6.4.3 Conclusions on transfer methods

Both the parallel and the concatenation methods yielded large reduction in transfer times for the tickets. Using GPRS the parallel approach is a clear winner, while the 3G tests showed a slight advantage for the concatenated files idea when using less than five threads. Since the gains were quite similar the less amount of changes required by the parallel method were the main reason for reaching the final decision, to implement the parallel file transfers. Additionally the gains when using GPRS were considered more important since no matter what improvement were implemented, the transfer times using a 3G device should be reasonably short.

The implementation initially utilised five parallel threads to do the transfer, but this was eventually changed to four. The reason for this is that several Motorola phones only allow four open sockets. Fortunately the difference should not be very large judging from the test performed.

All in all the usage of parallel file transfers should provide the application with transfer times as much as 50% shorter in the worst case scenario when five images needs to be fetched from the server.

6.5 Cookie monsters

During the first stages of testing the application interfaced with an internal development server in order to be able to test different features at will. A basic, but sufficient, session handling using cookies had been implemented and was functioning well. However once the application was tested using the live server the session handling unexplainably seized to work on some of the devices used for testing. Investigations

showed that the problem was isolated to devices using Telia as a carrier. Ultimately the problem was traced down to a proxy used by Telia which stripped all cookie information from HTTP responses on port 80. The only reply received from Telia when asked about removed cookie information was that this indeed is the intended behaviour by the proxy.

Since it can not be required from users to change the configuration in their devices and disable the proxy in order to run a single application, some other method of session handling had to be implemented. Fortunately the Java server by default also accepts the session id appended to the URL, meaning that in the end all that had to be done server side was a small addition to return the session id in the message body of the response to a successful login request. A few small changes in the application allowed the session id to be appended to the URL and the session handling was once again fully functional.

This solution should provide no apparent loss in security, mainly because the security in the application is quite low to start with. As mentioned earlier all communication with the server is unencrypted and there is no security gain or loss from having the session id in the URL instead of in a cookie.

In order to prevent similar problems with cookies using other carriers in the future it was decided that the cookie session handling should be scrapped and the URL rewrite method used instead.

6.6 Keypad mapping

The MIDP profile defines a series of default key codes for supporting devices. These includes the standard 0-9 keys, the asterisk (*) and the hash (#) keys. It is also practise to include two or more so called soft keys, there are regularly two of these soft keys located below the screen to the left and to the right. These soft keys have no default key code assigned to them and differs from phone model to phone model. The application uses these two soft keys to perform different user actions, but since there is no standard code for these keys a series of values has to be tested.

The different codes for the soft keys were gathered from different sources, including manufacturer data sheets and on device testing. The supported key codes should include most phones and some Pocket PCs. Unfortunately there is no way to guarantee support of all phones, especially not future ones, since the key codes can differ on future models. There is also the risk of “collisions” where the key code for a soft key of a supported device is the same as the key code for an other key on an unsupported device.

In addition to key codes there exist so called Game Actions. These are generated when the user operates the five-way joystick or pad (left, right, up, down and fire) and differ somewhat from the ordinary key codes. When receiving a key event the application should begin by checking if the received key code was a game action or not. If it was not the application can go on and handle it as an ordinary key code.

During testing on different mobile phones it was discovered that some Samsung phones trigger a fire action when one of the soft keys are pressed. This lead to a small addition

to the key parsing where the application also has to check if the fire action corresponds to a soft key or if it was indeed a proper fire action.

In order to maximise compatibility all the joystick/pad and soft key actions can also be performed using the 0-9 key pad of the phone. The reason for doing this is to make sure that even if some soft keys are mapped in a way that somehow makes the application unusable, it can still be operated by using the standard numeric keypad. Duplicating the functions to the key pad also makes it easier to use on phones where the joystick does not work very well, a problem often encountered by the author.

6.7 Exception extravaganza

While testing the application on a Motorola phone the threads doing http requests and transfers appeared to shut down inexplicable for no reason. This problem was traced down to the part of the networking code where connections were closed after the transfer had finished. Apparently the J2ME-implementation by Motorola sometimes throws an exception when closing the connection, something which was not supposed to happen according to the J2ME documentation. Once the problem was found it was an easy fix to simply catch any exception thrown when closing the connections, a solution which does not affect the operation on other phones at all.

Another rouge exception is caused by Sony Ericsson phones, where handling a key event may cause an exception. As mentioned before there are two types of key events, game actions and normal key codes. When checking an event to determine if it is a game action, the software throws an exception if it is not a game action but a normal key code. It was an easy fix to check for the exception, but once again it was not documented anywhere that the method in question could cause an exception.

Chapter 7

The resulting application

7.1 User Interface

The user interface was implemented using the Canvas class and runs in full screen mode. Colours and various small bits have been designed to hold similarities to the Skraplotto web page. In figure 7.1 some of the similarities can be seen, for example the rounded corners and the three small squares. It is believed that these small design features helps to strengthen the association between the mobile application and the web application.

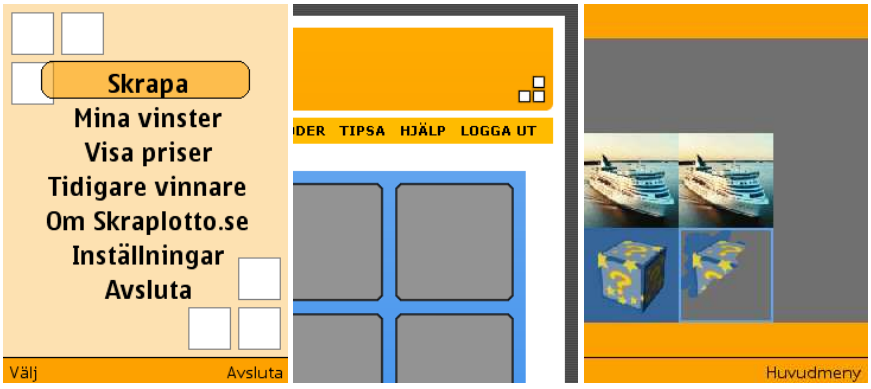

Figure 7.1: *The mobile application and a part of the web application side by side. Notice how design elements are reoccurring. The middle picture shows the web page while the other two are screen shots from the mobile application.*

In order to cater to the varying display sizes of mobile devices, the application can adjust some elements of the interface. Most of the images are automatically scaled to fit, but some elements can not be changed. An example is the available font sizes on different devices, which can be different from one device to another. The application tries to find the largest font possible still able to display the entire menu, but it is still possible that the fonts are too large in relation to the screen resolution to display all

the menu entries on screen. In this case the menus are changed slightly, and instead of a selector square moving over a static menu, the menu items moves up and down and can therefor also be off screen. An example of this is shown in figure 7.2 where the application is running on an emulated 128 by 128 pixel display.

Figure 7.2: *The application on a 128x128 pixel display. Note how some of the menu options are off screen.*

With the exception of one screen, the configuration screen, all of the application uses the canvas class as described above to display graphics. However the configuration screen utilises a form layout instead in order to be able to take advantage of features such as editable text fields and selection buttons. It was decided to use this layout instead of the canvas layout since the development of text entry would require a lot of time. Additionally the native text entry usually provides text prediction using T9 or a similar system, something which is not feasible to implement in a single application.

Because of using the form layout the looks of the configuration screen does not match the look of the rest of the application. As can be seen in figure 7.3 it also looks different depending on what device the application is being used on. The different look of the configuration screen is not necessarily considered to be a bad thing, since it allows the user to enter login name and password in a familiar environment.

Figure 7.3: *On the left is the configuration screen as seen using a Nokia emulator and on the right as seen on a Sony Ericsson emulator.*

7.2 Networking

As described in section 6.4.3 on page 25, the application relies on a threaded networking component which services the requests in the same order they were added to the queue. In reality this has been shown to work very well and improves the transfer times significantly compared to sequential file transfers.

When the application adds a request to the download queue, it can add it either as a cached or non-cached request. Non-cached requests are for changing data, such as tickets and other information. Cached data on the other hand are for data which should be static for any foreseeable time period, meaning the images associated with the campaigns.

When a request has been added to the queue the application will continue to poll the networking component to receive information about the download. The response contains information on how much of the download has been completed. As seen in figure 7.4, this information is used to display a percentage readout in a progress bar, keeping the user up to date on how the transfer is progressing and how much of the transfer still remains.

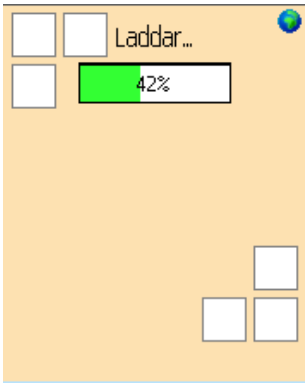

Figure 7.4: *The progress bar keeps the user informed on the current status of the file transfer.*

7.3 Data Caching

Both transfer times and the cost of transferring data over the air are aspects which have to be factored in when using applications which use over the air data transfers. In order to reduce time spent waiting for images to download and the associated costs the application uses a cache to minimise the amount of transfers necessary.

The image cache is of standard most recently used design. It keeps the last 20 displayed images in the non volatile memory of the phone and associates each image with a time stamp set to the last time it was accessed. When an image is needed the application first checks to see if the image is stored in the cache. If it is, the image is immediately returned, and if it is not it will be fetched from the server. Once fetched the image is

stored in a free space in the cache for later use. If the cache is full, the image with the oldest time stamp will be replaced.

The number of images to cache was chosen in order not to occupy too much storage space on the phone while at the same time store the images for as many of the active campaigns as possible. It was also taken into account that the people at Netix estimate the number of advertisement campaigns directed at a specific user should usually not exceed 20.

At the time of writing the average size of the images representing advertisements is somewhat less than 8 kbyte, leading to an approximate cache memory requirement of 160 kbyte. Since this is in no way an absolute number, the application is designed to check the available storage space at every start up. If the available space is less than 200 kbyte the application disables the caching system and also warns the user. The lack of available memory to activate caching will unfortunately lead to more transfers and longer waiting times for the user.

7.4 Error Handling

There is always a possibility that the information stored in the device persistant storage can become damaged, for example if the device runs out of power at the wrong time. If such a problem is encountered and the application can not make sense of the stored data, it will delete said data and re-initialise the relevant part of the storage the same way as it does the first time after being installed on a device.

If an error is detected in the XML transfers, the application will notify the user of the problem and wait for the user to shut down the application. The same thing happens if there is a problem with the network connection, as seen in figure 7.5, the application shows any available error message from the Java engine and waits for the user to end the application.

Any unexpected terminations of the application should be handled by the Java implementation, automatically cleaning up any used memory and other resources.

Figure 7.5: *Error message concerning a faulty URL.*

7.5 Compatibility

The application has been tested on a series of different phone models from several manufacturers. Using J2ME to implement the application has allowed it to be used not only on the targeted Java-compatible phones, but also on Symbian phones and Pocket PCs which include a J2ME environment.

Some small changes were implemented in order to support a couple of devices, but no major changes has been necessary in order to support any specific device. In the end, the compatibility is considered to be good and it is believed that the application should work well even on most untested devices. A chart of the different devices used for testing and some comments on any peculiarities experienced can be found in Appendix D.

The only devices which proved to be a real problem to support were the two LG phones, partly because references for the Java implementation on these devices could not be found on the manufacturer website. Unfortunately the lack of documentation and the encountered bugs made it impossible to modify the application to function correctly on these devices.

Regarding compatibility with the existing server platform and software the application has fared very well. Only a couple of small additions to the server software were required in order to get the current functionality. These small additions could be applied in minutes, without any impact to the current web-based front end.

7.6 Deployment

The application is considered stable and ready for deployment. It has also been tested by several unofficial beta testers. Despite this it is uncertain if it will be deployed to the public or not.

Currently the advertisers provides three pictures of different sizes to promote their product or services. Thereby they know exactly how the advertisement will appear on a users computer monitor. On a mobile phone on the other hand, the presentation of the advertisement will differ between different devices with different screen resolutions. On a device with a very low resolution, in a worst case scenario, the advertisement might not even be legible. On the other extreme a device with a very large screen, such as a hand held computer, might make the images appear blocky and poorly re-sized.

Not being able to promise how the advertisement will appear to the user could make it harder for Netix to get companies to buy advertisement space from them. Eventually this problem will hopefully be resolved and there might even appear campaigns especially targeting mobile phone users.

In addition to the problem with the presentation of the advertisements, the built in web browser is also a limiting factor. Most of the advertisers do not have a web site designed for mobile phones, and the appearance of an ordinary web page can be very lacking when viewed with a low end phone. Since much of the Skraplotto revenue at the moment comes from users visiting the web pages of the advertisers, it is important that the web page of the advertiser can actually be viewed. This is most

likely a problem which will solve itself over time as the browsers become more and more capable at the same time as companies increase their presence on the mobile web.

Chapter 8

Evaluation

8.1 Conclusions

I believe I have reached the goal which had been set for this project, to design and implement a ready to deploy mobile application. Unfortunately it is not sure if the application will be made available to the public or not. While this feels like a bit of a disappointment I am still very satisfied with the outcome of the project and I feel that I have had the opportunity to assimilate a lot of new information and knowledge during my work.

8.2 Additional thoughts

I choose to do do this project for a couple of different reasons. The main reason is that I have always been interested in programming for devices which have limited processing power and storage capabilities. Today most computers have enough raw power to solve many problems by brute force. On the other hand, while programming for for example mobile phones one has to take into consideration the many limitations of the device in question, a challenge I appreciate.

Another reason for me choosing this project is that I believe that the number of applications for mobile devices will increase significantly during the next year or two. Having some experience in programming for these devices might just come in handy in the future.

8.3 Future work

The main aspect of the application needing continued development is that to expand the range of tested and supported devices. Cost limitations prohibited me from testing the functionality on as many phone models as I would have wanted to. It would surprise me if additional testing did not reveal incompatibilities with other devices.

One thing I would like to do differently in the application, should I get a chance to, is to use a smaller and more compact XML parser. Not that there is anything wrong with the one currently in use, but I suspect the application could have been made a bit more compact. Changing the inner workings of the client means that a lot of compatibility testing would need to be performed again, a task I unfortunately did not have the possibility to do during the end of this project.

An interesting additional feature to implement would be the possibility to scratch tickets using a touch screen, something which is becoming increasingly common among mobile devices. This option was not investigated at all during the development due to cost restrictions as no phone with a touch screen was available. This might be a feature for a second version of the client, if one should be developed somewhere in the future.

Bibliography

[1] Adobe Systems Incorporated, "Supported Devices: Handset Manufacturers", *www.adobe.com*, 2007. [Online]. Available: http://www.adobe.com/mobile/supported_devices/handsets.html [Accessed: February 16, 2007].

[2] Sun Microsystems, *JSR-000030 J2ME(TM) Connected, Limited Device Configuration Specification 1.0a Final Release*, May 19, 2000

[3] Sun Microsystems, *J2ME(TM) Connected Limited Device Configuration (CLDC) Specification 1.1 Final Release*, March 2003

[4] Sun Microsystems, *J2ME(TM) MIDP Specification 1.0a Final Release*, December 15, 2000

[5] Sun Microsystems, *JSR-000118 Mobile Information Device Profile Specification 2.0 Final Release*, November 5, 2002

[6] Symbian, "Symbian OS phones", *www.symbian.com*, 2007. [Online]. Available: http://www.symbian.com/phones/index.html [Accessed: February 14 2007]

[7] Jonathan Knudsen, "Parsing XML in J2ME", *Sun Developer Network*, March 7, 2002. [Online]. Available: http://developers.sun.com/techtopics/mobility/midp/articles/parsingxml/ [Accessed May 15, 2007].

[8] "Sparta", [Online]. Available: http://sparta-xml.sourceforge.net/ [Accessed May 25, 2007].

[9] Rajiv Chakravorty, Joel Cartwright and Ian Pratt, "Practical Experience with TCP over GPRS", in *Global Telecommunications Conference*, 2002, pp 1678-1682.

[10] James F. Kurose and Keith W. Ross, *Computer Networking: A top-down approach featuring the Internet*, Addison Wesley, 2001, pp. 249-260

Appendix A

GPRS response times

The following charts shows the results from a series of data transfer tests made using GPRS for data transport.

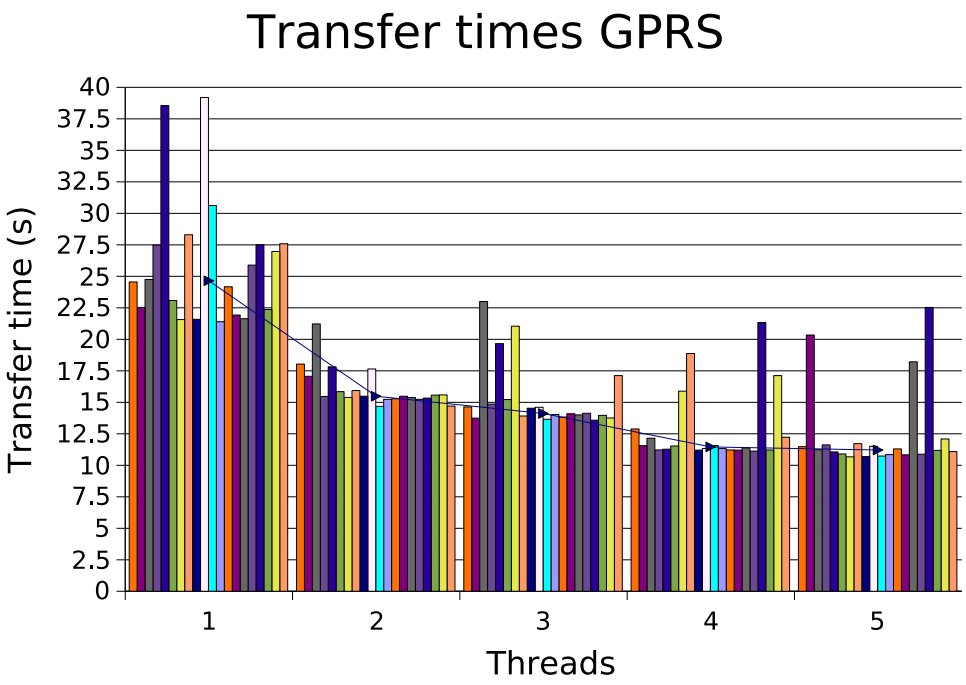

Figure A.1: *Time needed to transfer a total of five 7500 byte data files using one to five parallel transfer threads.*

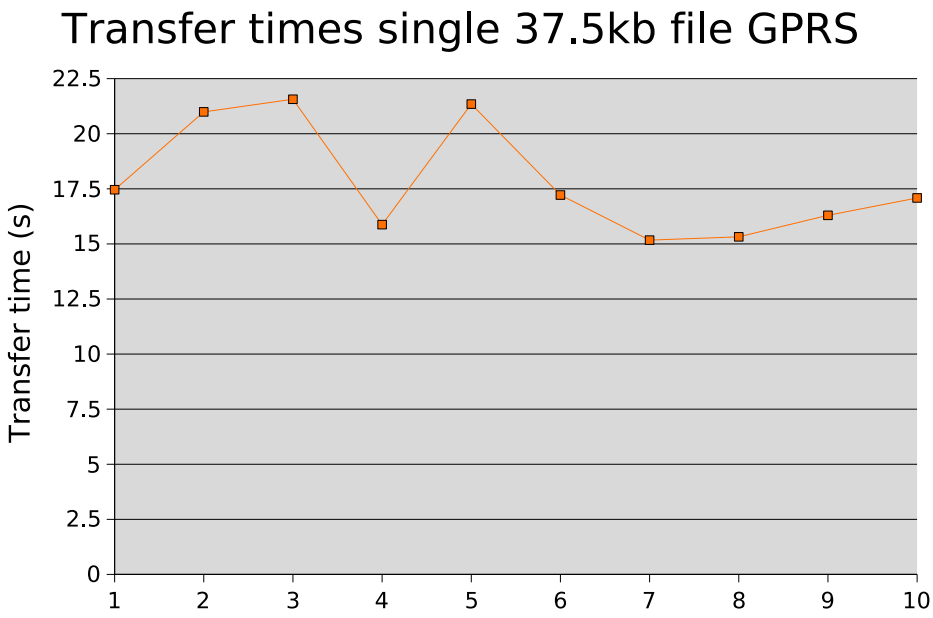

Figure A.2: *Tests of the time needed to transfer a single 37500 byte data file, performed 10 times.*

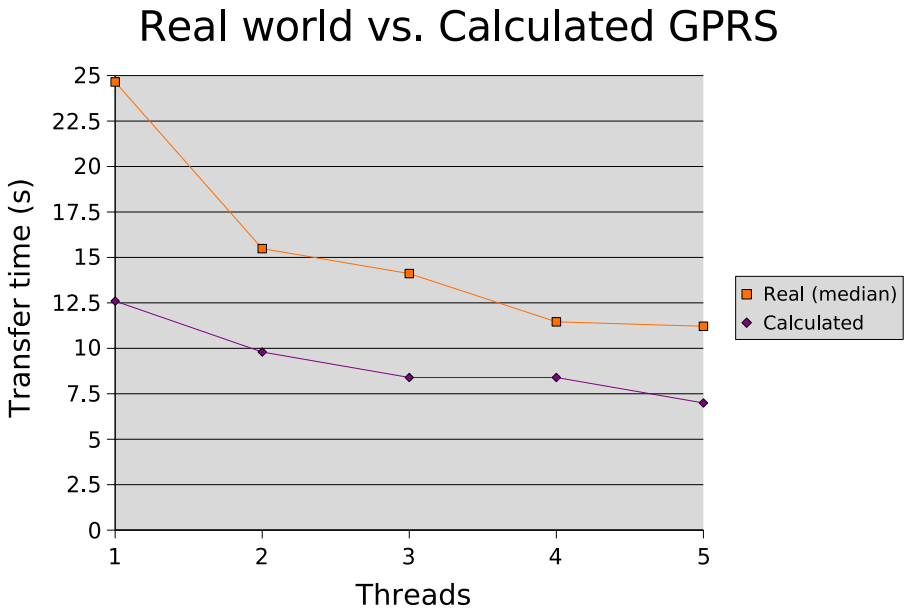

Figure A.3: *Comparison of calculated best case and real world results for transferring five 7500 byte files using one to five parallel transfer threads.*

Appendix B

3G response times

The following charts shows the results from a series of data transfer tests made using 3G/UMTS for data transport.

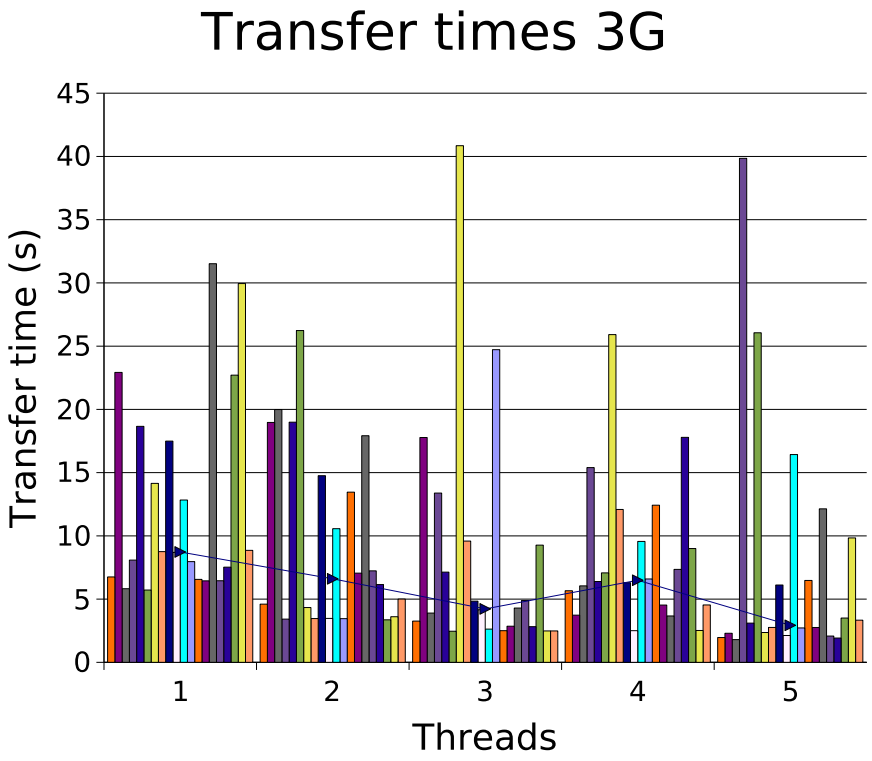

Figure B.1: *Time needed to transfer a total of five 7500 byte data files using one to five parallel transfer threads.*

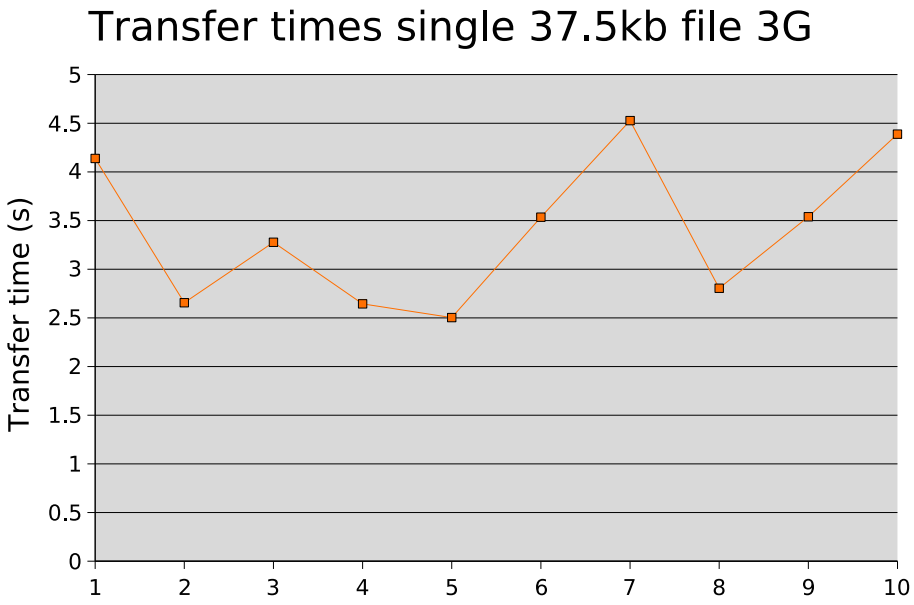

Figure B.2: *Tests of the time needed to transfer a single 37500 byte data file, performed 10 times.*

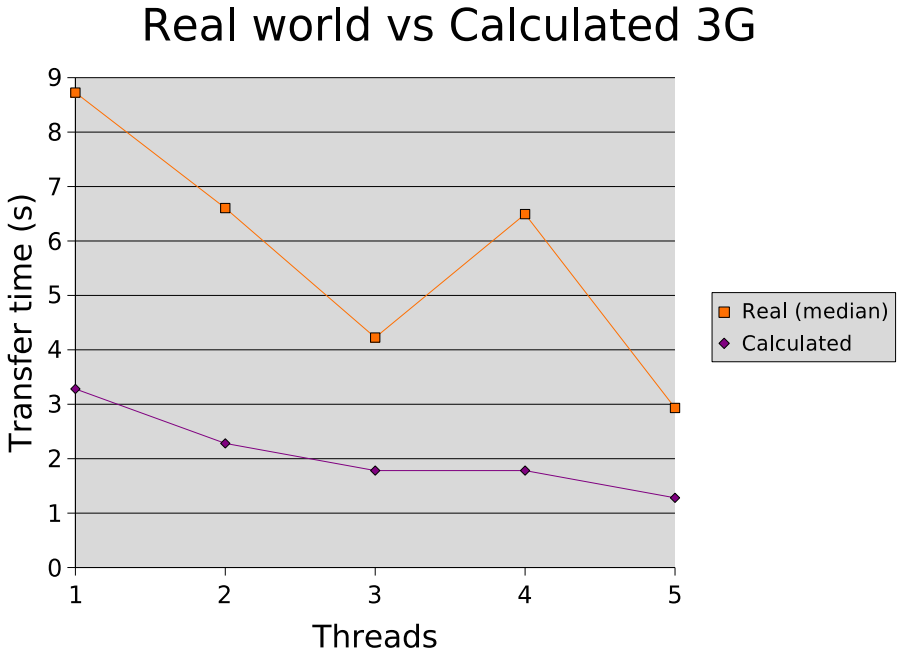

Figure B.3: *Comparison of calculated best case and real world results for transferring five 7500 byte files using one to five parallel transfer threads.*

Appendix C

User interface concepts

C.1 Mock-ups

Figure C.1 to C.4 shows early concepts of the user interface, constructed in Open Office Draw using images from the existing Skraplotto web page.

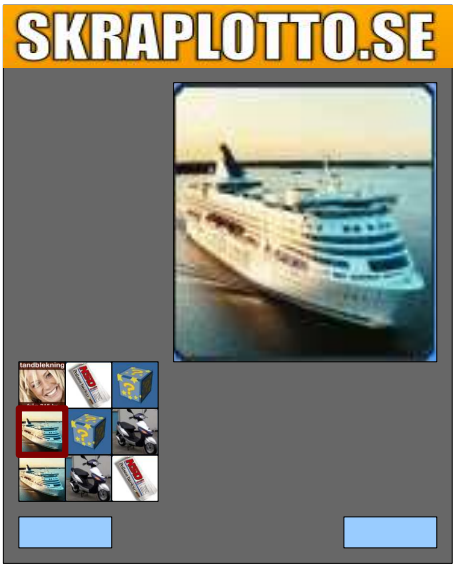

Figure C.1: *Big view of the active square with a miniature view of all squares arranged in a three by three pattern.*

Figure C.2: *"Fish eye" view of a three by three pattern with the active square magnified and a miniature view of the remaining eight.*

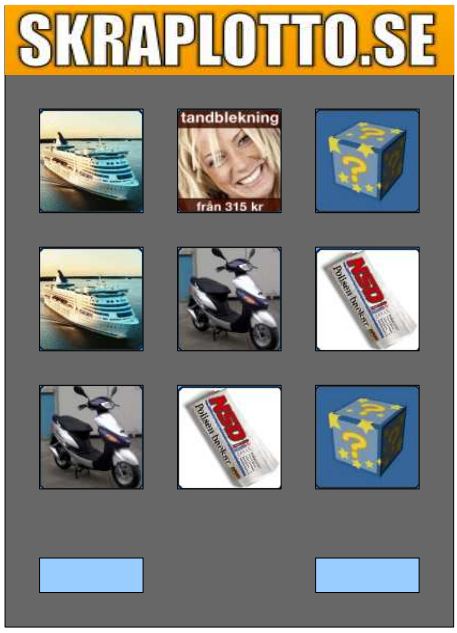

Figure C.3: *Classic three by three view with all square static sizes and positions.*

Figure C.4: *Big view of the active square with a miniature view of all squares arranged in a line.*

C.2 J2ME Demo

Screen shots from the user interface demo application running on an emulated Sony Ericsson phone are shown in figure C.5 to C.8. Some minor changes has been made compared to the earlier mock-ups, mainly to simplify implementation.

Figure C.5: *Big view of the active square with a miniature view of all squares arranged in a three by three pattern.*

Figure C.6: *"Fish eye" view of a three by three pattern with the active square magnified and a miniature view of the remaining eight. The layout of the squares changes dynamically when the user selects different squares.*

Figure C.7: *Classic three by three view with all square having static sizes and positions. The square sizes has been increased to cover the entire screen are.*

Figure C.8: *Big view of the active square with a miniature view of all squares arranged in a line.*

Appendix D

Tested devices

This table shows the devices on which the application has been tested. There are also comments regarding different quirks discovered and how they were taken into account when developing the application.

Device	Works	Remaining problems	Comments and fixes
HTC Windows Mobile	90%	The application does not receive notification on some screen changes such as when hidden or restored.	Odd softkey key codes.
LG KE970	40%	Platformrequest to open native browser does not work, connections die for no reason.	No workarounds found - barely supported.
LG U880	5%	Very little works. Connections die for no reason, platformrequests and certain Java operations causes the phone to lock up, etc.	Not feasible to support - practically unsupported. Very buggy device.
Motorola c380	100%		Allows for a maximum of 4 open network sockets.
Motorola PEBL	100%		Throws exceptions when network connections are closed.
Nokia 6131 (Series 40)	99%	Platformrequest to open native browser needs to close down the application. This is a known limitation for some phones and described in the J2ME documentation.	Disabled the possibility to invoke native browser.
Nokia 6680 (Series 60)	100%		Had to implement a fix for transparency to work as intended.
Nokia N93 (Series 60)	100%		Workaround for getRGB and transparency implemented.
Nokia N95 (Series 60)	100%		
Samsung SGH-E530	100%		Softkeys triggers fire events.
SE J300i	100%		
SE K700i	100%		Game event parsing throws exceptions. Probably common for Sony Ericsson phones.
SE K800i	100%		
SE K810i	100%		