Feasibility of port from Android to Blackberry

- Möjligheten att porta från Android till Blackberry

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Computer Engineering

Bachelor thesis:
Joen Jönsson

© Copyright Joen Jönsson

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2011

Abstract

The mBlox Device Software, MDS, is a library that developers integrate into their applications to expose an API on the internet to access resources on a mobile device, such as an Android smart-phone. This report discusses the possibility to port the existing implementation of the mBlox Device Service from the Android Platform to the Blackberry Platform.
In the time that was given a very much working version of the MDS was implemented, though it is not fully tested a lot of the functionality do work, at least when all parameters are correct. This is what a proof of concept solution is all about.
The conclusion is that since both platforms are using Java for their applications a port of an application without a user interface is fairly easy, and because both platforms support multitasking the design can stay roughly the same. The main difference for applications or libraries without an UI, like the MDS, is the differences between Java Micro Edition and Java Standard Edition. However if an application does have an UI the differences are more noticeable, a developer should follow the UI guidelines for the particular platform and they are different on both platforms.
All the goals that were set up was fulfilled.

Keywords: Blackberry, Android, Port

Sammanfattning

mBlox Device Software, MDS, är ett bibliotek som utvecklare inkorporerar i sina applikationer för att synliggöra ett API på internet som gör det möjligt att komma åt resurser från en telefon, till exempel en Android telefon. Rapporten diskuterar möjligheten att porta mBlox Device Software från Androidplatformen till Blackberryplatformen.
Av den tiden som gavs blev en fungerande version av MDS implementerad, även om den inte är fullt testad fungerar den om alla parametrar är korrekta. Det är detta en Proof of Concept lösning handlar om.
Slutsatsen är att eftersom båda bygger på Javaplatformen är en portning av ett system utan ett grafiskt gränssnitt relativt enkelt och eftersom båda plattformarna stödjer multikörning kan designad vara väldigt lik. Huvudskillnaden för en applikation eller ett bibliotek utan ett grafiskt gränssnitt, som MDS, är skillnaden mellan Java Micro Edition och Java Standard Edition. Ska en applikation däremot ha ett grafiskt gränssnitt så blir skillnaderna mer påtagliga. En utvecklare ska alltid försöka följa riktlinjerna som finns om hur man designar ett gränssnitt för just den plattformar.
Alla mål som sattes upp nåddes.

Nyckelord: Blackberry, Android, Portning

Foreword

I would like to thank the guys at Mashmobile who helped me complete this bachelor thesis. I would also like to thank Christian for supervising the thesis.

List of contents

1 Introduction

1.1 Background

The mBlox Device Software, MDS, is a piece of software that is included into a mobile device application and it takes care of the communication between the device and the mBlox network, Mnet (internal network to handle communications with the phone), and thereby exposing an API, Application Programming Interface, on the Internet to access resources on the device. Web developers can then develop applications and services that access these resources in the same way independently of what operating system the device is using. This makes it much easier for the developers because they don't have to research on how to do all things, like connecting and encryption on each platform and this speeds up the development. This is the power of the MDS. An example of an application that uses the MDS library could be an application that handles bookings for events. When there is a new event available the web application pushes a message to the device that downloads information about the event and then lets the user see that there is a new event available.

MDS is available on the Android, Symbian and the iOS platforms. Blackberry is seen as an important platform to broaden the install base for MDS. The product is aimed for third party developers, mostly other companies as it requires a subscription to use the mBlox infrastructure.

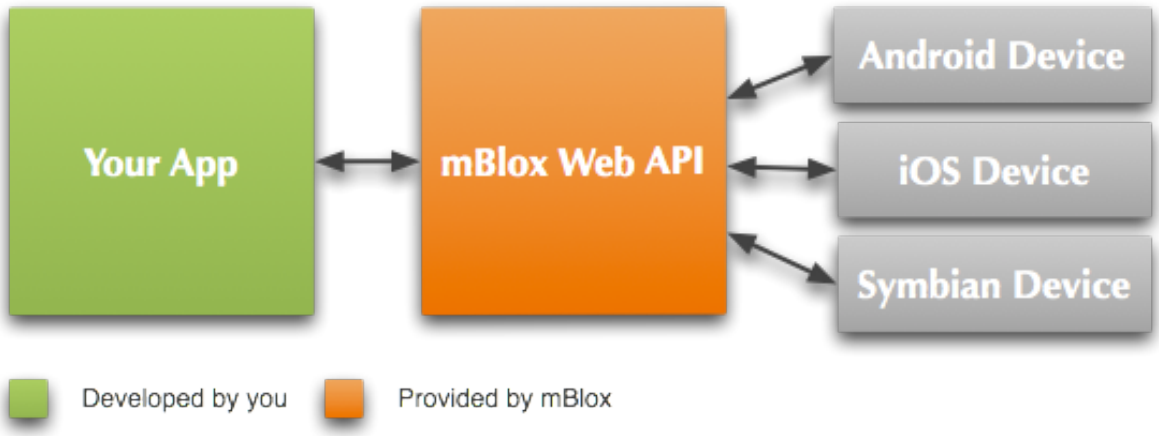

Figure 1 Schematic of the current MDS system. The plan is that Blackberry Device should be added to the picture. (Picture from Mashmobile Developer site [5])

1.2 Mashmobile

The employer of this thesis is Mashmobile, it is a company in Lund owned by mBlox. There is about 20 people working in the Lund office and they are stationed at the Ideon Science Park in Lund.

Mashmobile offers a solution of a push system that is used in the same way on multiple platforms. This means that developers can develop an application running on a web-server who instantiate a push-message to the Mashmobile network, referred to as the mBlox network later in the report. The Mashmobile Network then sends out that push-message to devices, independently of what operating system that is used on the device.
mBlox is a company that specialises in infrastructure that allows companies to send SMS to their customers, like advertisement campaigns or information messages, to a large number of telephones.

1.3 Purpose and Goal

The goal is to study the current MDS implementations and then study the Blackberry SDK to see how the MDS can be designed and implemented on the Blackberry platform. If there is time available then a proof of concept software is to be implemented.
As Blackberry is such a large platform it is seen as a potential candidate to expand the MDS installation base and to reach more customers.

1.4 Problem Formulation

Today the mBlox Device Software is available on the Android, the iOS and the Symbian platforms. As Blackberry is one of the most used operating systems for Mobile Smart-phones it is seen as an important platform to broaden the install base for MDS.

Is it feasible to port the Android version of MDS to the Blackberry Platform?

Here are some of the areas that might become problematic and which needs a more thorough investigation.

- Secure communications.
  Because the communication with the mBlox Network is via the Internet people might listen to what we have to say. To prevent this the mBlox network uses TLS as its transport encryption. This means that a working version of the MDS for Blackberry needs to support TLS in order for the mBlox Network to accept the communications.
- Connectivity.
  As stated above, the communication with the mBlox Network is via the Internet and as such the MDS needs to be able to communicate via the internet. An investigation on how that is done on the Blackberry platform is needed to get a working version of the MDS.
- Background applications.

It is important to be able to maintain a connection to the mBlox Network even if the application is closed, otherwise the device would not be able to receive any messages from the mBlox Network when the user exits the application. This needs some investigation to if and how that is possible to achieve on the Blackberry Platform.

- Blackberry Push
  The Blackberry push is a service provided by Research in Motion, the developers of the Blackberry platform, which lets developers send messages to devices. A study must be made to see how the Blackberry Push Service can be incorporated into the MDS.

1.5 Limitation

The implementation of the Blackberry version of MDS is just a proof of concept solution. While the aim of the port is to examine how the MDS can be designed and implemented on the Blackberry platform not everything might be optimally implemented for the platform.

1.6 Method of work

The method of work was not any typical project model, like SCRUM or XP. Because I worked alone I had the freedom to do whatever I wanted and as such I spent a lot of time looking at the Android code, drawing up models of how it works and trying to find a way that was doable on the Blackberry Platform. An overview of the work was that the first weeks where spent learning how the MDS works, learning the Mnet structure and learning how the Blackberry platform is designed. After that the porting began, with an analysis of each section on the Android platform and an analysis of how it would work on the Blackberry platform.

1.7 The report

This report will describe how the Blackberry platform works, how the MDS works and what kind of issues a developer might face when trying to port an existing application from Android to Blackberry.

2 Background

In this chapter the systems involved will be presented.

2.1 Blackberry

The thesis is about the feasibility of porting an existing system for Android to the Blackberry platform, to do this an introduction to the Blackberry platform is needed. This part will describe the Blackberry platform in short.

2.1.1 Description

The Blackberry platform is a mobile device platform developed by Research in Motion, RIM. They develop both the closed source operating system, Blackberry OS, and the devices on which the OS runs. It was originally designed to be used by enterprises so they could read emails and check their calendar on the device. With the help of the Blackberry infrastructure enterprises were able to set up servers that pushed out (explained in chapter 2.5) new email, calendar events and tasks to the devices. Most of the newer Blackberry devices have most of the standard functions of a modern smart phone like GPS, Wi-Fi and camera. One feature that might interest the ordinary customer is Blackberry Messenger, which is a messaging application, much like MSN or Google Talk, the difference is that all messages goes through the Blackberry infrastructure and it is free to use for anyone with a Blackberry device and a Blackberry Data Plan. Other than this the Blackberry platform does not offer anything exceptional for the ordinary customer. For the enterprise customer there are many features that might be interesting. More on that in chapter 2.1.3.
On the Blackberry platform all applications are written in Java, they are built upon the Java Micro Edition.
As Blackberry is a closed source platform much of the technology that is used to build the platform is unknown for the public. This means that the major technical information is known only to the developers at RIM.

2.1.2 Integration

One of the advantages of the Blackberry Platform is that it lets developers integrate their applications with the applications that Blackberry provide. For instance a developer can access the messages application which lets them add messages to the device which will let the user get messages in the same application, independent of what application sent the message. This makes the developer able to develop applications that really feel like a natural part of the operating system.

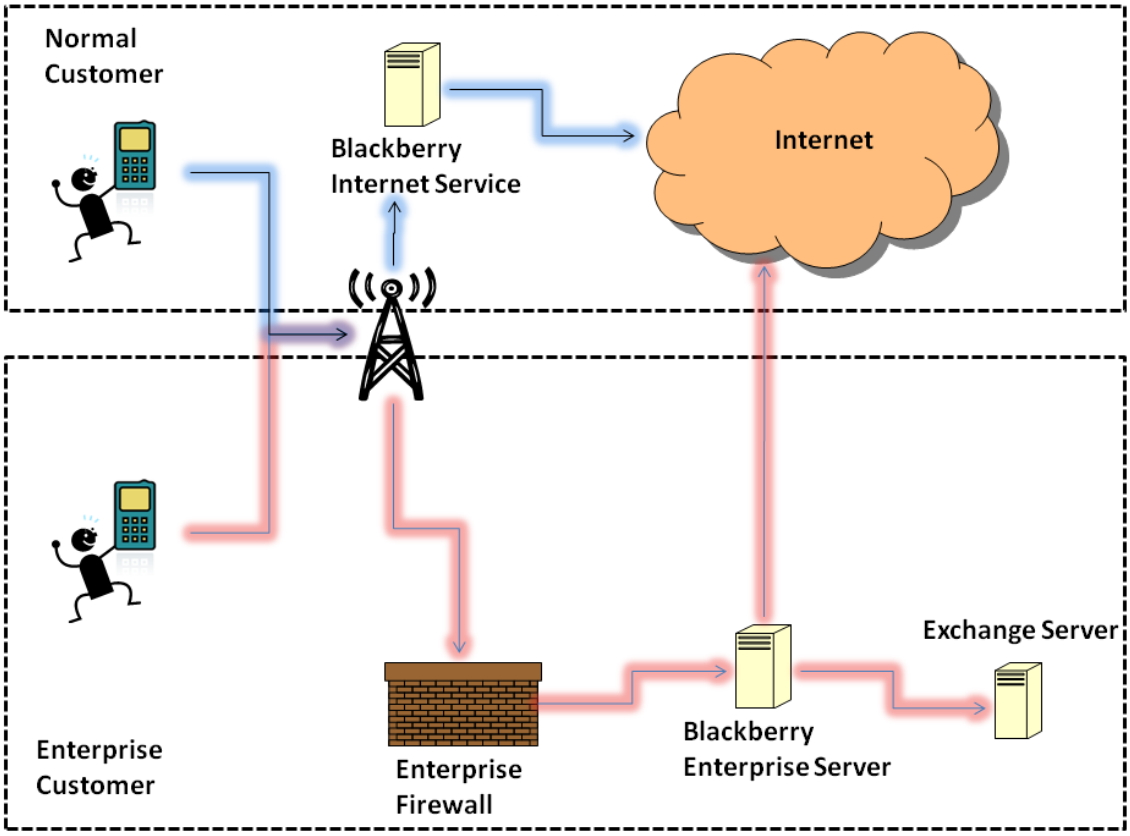

Figure 2 Shows the Blackberry Infrastructure from an Enterprise Customers and a Regular Customers view.

2.1.3 Blackberry Enterprise Server

The Blackberry Enterprise Server, BES, is a local server that enterprises can use to connect their Blackberry Devices to [8]. The BES connects to existing email services, like Microsoft Exchange or IBM Lotus Domino, and pushes emails and calendar information to the devices. The BES also has good security features which lets an administrator to set what kind of applications a device can use, the administrator can even wipe all information on the device remotely if a device is lost. This is one of the features that makes the Blackberry platform so widely used in corporations across the world.

2.1.4 Blackberry Internet Service

The Blackberry Internet Service, BIS, is a kind of Enterprise Server but for non enterprise customers. It lets a user connect up to 10 email accounts and have the Blackberry Internet Service push new messages to the device.
To use the Blackberry Internet Service a special data plan is needed from the wireless service provider.

2.1.5 Blackberry PIN

All Blackberry devices have a unique 8 digit hexadecimal pin number. These are used as unique identifiers for the devices and can be used to send messages between devices and for the BIS/BES to address push messages.

2.2 mBlox Device Software

This part will describe the MDS in short.

2.2.1 Description

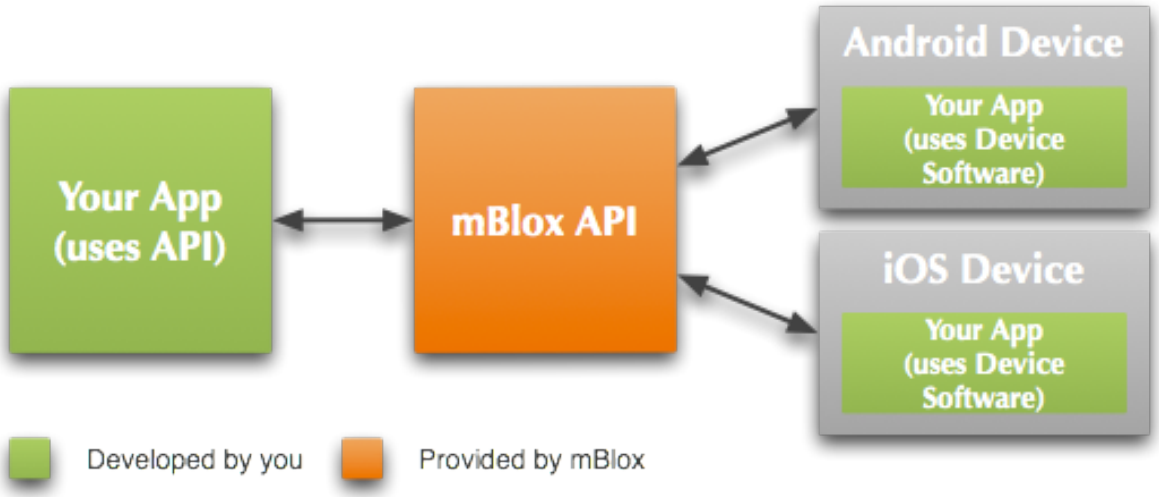

Figure 3 An overview of the current MDS and the mBlox API. The plan is that Blackberry Device should be added to the picture. (Picture from Mashmobile Developer site [5])

Essentially MDS is a library that is included into mobile device applications and the MDS will then take care of communications with the mBlox Network and thereby exposing an API on Internet to access resources on the device. Web developers can then develop applications and services on the web that access resources on devices in the same way independently of operating systems running on the device. As seen in Figure 2 the developer only needs to provide with an application for the web (the left square) and one or more applications for the devices. The mBlox API handles the communication between them with an easy API for the developer. In the example with the events manager, the developer would develop an application that runs on a web server (provided by the developer). The web application communicates with the mBlox API telling it when a new event has been added. The mBlox API then sends out a message to the registered devices to notify the user about the new event. Of course the developer needs to implement an application for each platform that are to be supported, but the simple API from mBlox makes the development easy and the web application can be exactly the same. This is the advantage of the mBlox Device Service.
More information about the MDS is found in chapter 3.

2.3 mBlox Web API

The mBlox web API is the API developers use when their web applications need to connect to devices. This API is what communicates with the MDS on the devices and it is this API that third party developers will use to address devices and to get resources of the devices. This runs on the mBlox servers so

there is no need to look into this for this thesis as the communication between the MDS and the Web API is the same independent of what platform the MDS runs on.

2.4 Blackberry and Android

Since a study will be made of the already existing Android platform implementation of the MDS the following chapter will explain the Java Standard Edition, which Android uses, and the Java Micro Edition, which Blackberry uses.

2.4.1 Java Micro Edition

The Java Micro Edition, Java ME, is a special version of the Java platform designed to be used on devices with low memory and CPU capacity. It is based on the Java 1.3 platform.

2.4.2 Java Standard Edition

The Standard Edition of the Java platform, Java SE, is normally used for developing Java applications to a desktop. The current version (May 2011) is 1.6.

2.4.3 Difference between Java ME and Java SE

One of the differences between the Java Micro Edition platform and the Java Standard Edition platform is that the Standard Edition has a lot more libraries implemented than the Micro Edition. The entire concurrent package and most of the utilities package is missing from the Java ME libraries. This means that a developer who wishes to use those libraries will have to implement and test them themselves, this makes developing for the Micro Edition slower, especially if the developer is more used to the richer libraries of the Standard Edition.

One example of this is that a commonly used code snippet:

```
ArrayList<String> list = new ArrayList<String>();
```

It can not be done in Java ME because the class ArrayList isn't implemented. Also the use of generics, <String> in the example above, wasn't introduced in Java until version 1.5 which means that it isn't available in Java ME. This means that all lists contains arbitrary objects which has to be cast into the preferred object type when used. This means that there is no check when a developer puts some data into the list to what type of object it is. If a developer has control this should not be any issues, but as the MDS is a library designed to be used by third party developers it has to be specified clearly in order to not cause confusion.
The `Iterator` is not implemented in Java ME either, for the standard `Vector` class, which is similar to the `ArrayList`, there is a `elements()`

method which returns an `Enumeration` which contains the elements. The `Enumeration` class is a bit more clumsy to use than the more refined `Iterator`. Also RIM encourages the developers to not overuse the `Enumeration` because it can be slow in certain situations.

2.4.4 Android API

The Android platform uses the Java SE platform plus additional libraries for its applications. The extra libraries include packages such as `android.os` and `android.view`. The `android.os` provides basic operating system functions, like inter-process communication, and `android.view` provides basic user interface classes. [6]

2.4.5 Blackberry API

The Blackberry platform uses the Java ME platform plus additional libraries provided by RIM for its applications. As with Android these packages handles basic operating system functions and user interface capabilities.

2.5 Push Technology

Push technology is a technology where a server sends a message to a client when there is new content, instead of the client polling, asking, the server with fixed intervals. This technology makes a client get the data at the same moment it is made available on the server instead of the possible waiting time when using poll technology.

One common way to implement push technology is to open a connection to a server and keep it open. When reading from a stream with no content, at least in Java, the thread will wait until there is any data available and if the server waits to send data until it becomes available it can be used as a push message. A lot of common push applications use this solution, like the Apple Push Notification Service [9] and Push email [10].

Both Android and Blackberry has build in support for their respective push-technology. In Blackberry a developer can incorporate the Blackberry Push Service into their applications to send push-messages to the phones through the Blackberry Infrastructure. In Android the technology is called C2DM and a developer can incorporate that into their application to send push-messages to the devices via the Google Infrastructure. The problem is that all systems have different solutions so a developer needs to learn them all if they are to be used. This can be a bit tricky and would take a lot of time for a developer to adapt their web-application to handle all the different solutions provided by each platform.

3 Study of the mBlox Device Software

The MDS is, as mentioned before, a library that a developer includes into their application to expose an API on the internet so that a web application can access resources on the device. It is available for the platforms Symbian, iOS and Android. Because the Blackberry platform is so similar to the Android platform a study will be done on the Android implemented version.

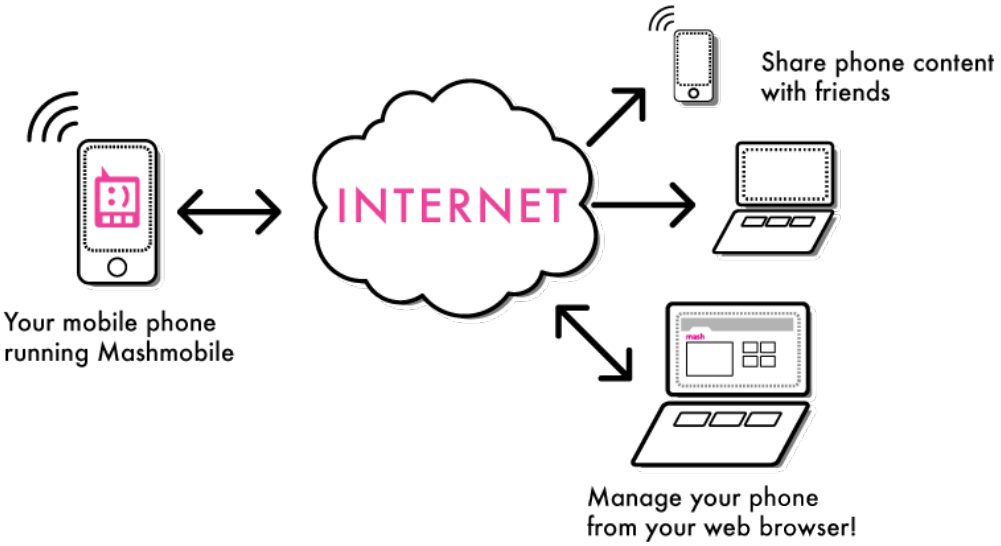

Figure 4 This shows some uses of the MDS. (Picture from Mashmobile Developer site [5])

3.1 An application using MDS

An example of an application using MDS could be a store, which sells bikes. They could have an Android and an iPhone application using the MDS. When the store get a new special offer they can send it out the phones which has this application installed. In this way they can send out special offers to people who want the special offer.
They could also add functionalities like competitions, like a photo competition. Take a picture where a user rides a bike in an extreme way and then the MDS could send that picture to the stores web server automatically. The store can then decide who has the best picture and give out the price. All this can be done with minimal effort from the store because the MDS handles the things like sending out the message with the competition rules to the devices, and when a user takes the photo it will automatically upload the photo to the store. All this can easily be done with devices that are running multiple operating systems. This is one advantage of the MDS.

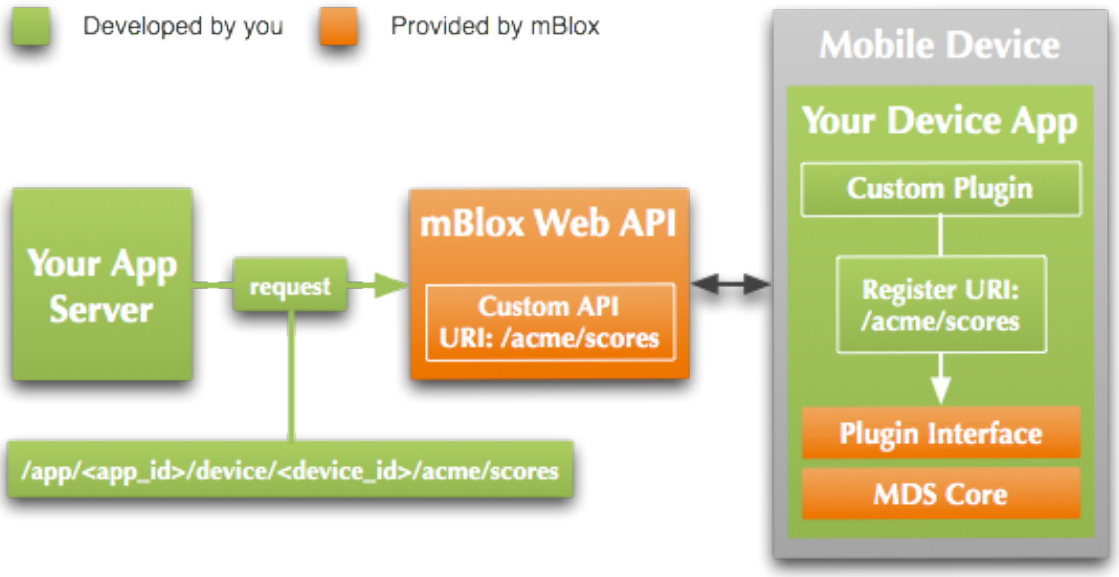

Figure 5 An overview of the flow of communication between a web application and the MDS. (Picture from Mashmobile Developer site [5])

3.2 RichPush Plug-in

This is a plug-in for the MDS that is used to push messages to the phone that should be activated at a certain time, or in a certain location. It can be used to let customers know about certain special offers in certain places or to notify them about an event that is starting at a certain time. This plug-in also lets developers analyze the push messages with aspects like viewing time of messages and number of devices who downloaded and read a message.

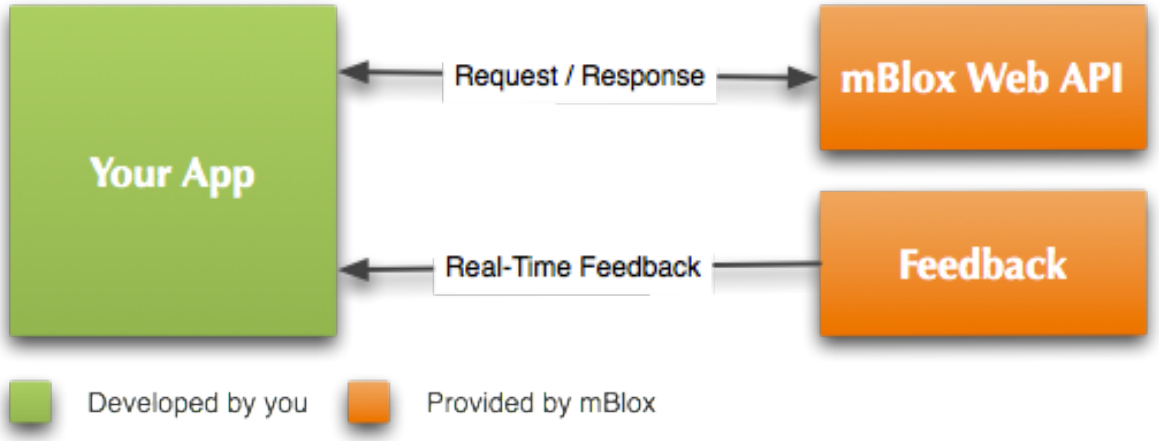

Figure 6 Shows how a developer gets real-time feedback. (Picture from Mashmobile Developer site [5])

3.3 Android version

Both Android and Blackberry applications are written in Java, which means a lot of the code, can be reused with the same design and only slight modifications to the syntax. The MDS library handles the connection to Mnet,

it also handles everything that has to do with security and encryption, like storing certificates.

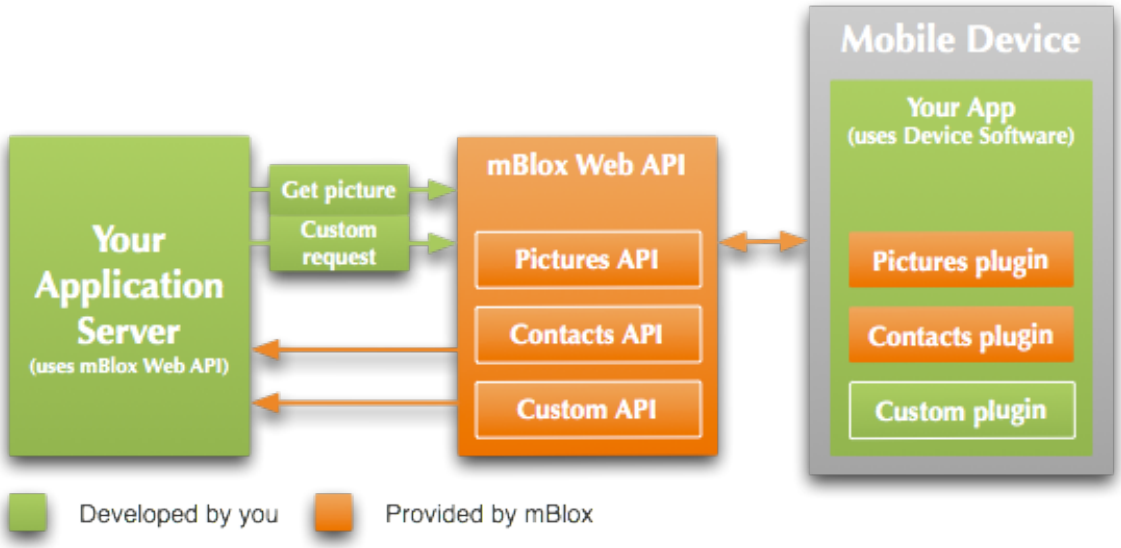

Figure 6 An overview of the flow of communication between a web application and the MDS. (Picture from Mashmobile Developer site [5])

The MDS is designed to be simple for a third party developer to get started using the MDS, and has extensive documentation on the mBlox developer website. The MDS library has a number of pre-made plug-ins that a developer can use in their application. Example of plug-ins are plug-ins that offer the ability to manage Contacts on the device, manage files on the device and get the location of the device. A developer can also develop their own plug-ins, with the help of an easy and powerful interface. This enables developers to create their own, very specific, and very powerful plug-ins that fit their purpose exactly. The advantage of the MDS is that the web interface can be the same independent of what OS the device is using, as long as that OS is supported by MDS.

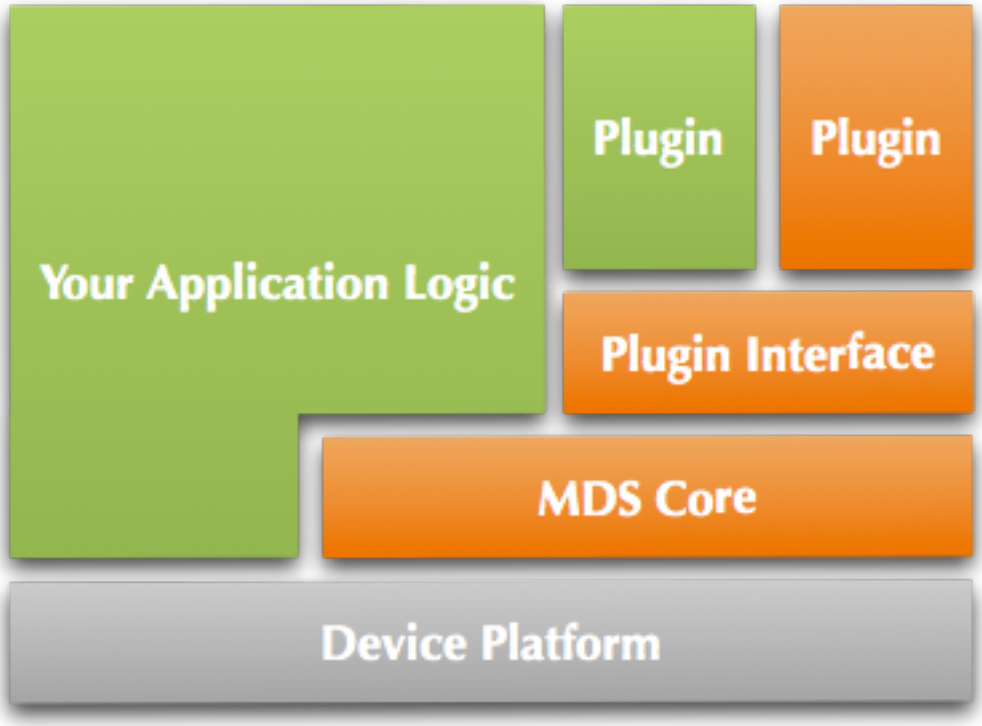

Figure 7 Schematic figure of how a mobile application use the MDS. (Picture from Mashmobile Developer site [5])

Things usable from the Android version with minimum changes were the Connectivity logic, with the different registration/connection phases and the retry logic since most of the code is pure java. Also the encoders and decoders for the Mashmobile Network Protocol, which is the protocol used for communicating between the Mnet and the MDS designed specifically for this system by Mashmobile, could be reused without much change. Things that need to be designed specifically for the Blackberry platform includes setting up the sockets for communication and setting up security for the communication. Also everything that saves data to the device needs some changes to work on the Blackberry platform because the two platforms have different ways of saving data. The Android version allows developers to store data in, what is called, the Shared Preferences, in the same way as the Blackberry version allows developers to store data in Persistent Objects. They are quite similar in functionality, the Shared Preferences in Android lets a developer store the data types: Boolean, Float, Integer, Long, String. The Persistent objects in Blackberry lets the developer wrap any of the data types in other objects, so a developer can store objects, like lists and such. This is a bit more powerful than the Android version.

4 Study of the Blackberry platform

The following chapter is a deeper study of the Blackberry platform and will address some of the main concerns of this port and the possible solution/s.

4.1 Background application

4.1.1 The problem

For the MDS to work it would need to handle background processes, like keeping a connection alive while being able to use other applications. The connection should not be lost every time another application is used. That would require a lot of data to be unnecessarily sent in the connection phase because the TLS handshake sends quite big keys back and forth, also the Mnet sends quite a lot of data upon login.

4.1.2 Solution

Luckily the Blackberry Platform supports applications to be run in the background, in fact, RIM encourages the developers to have their applications running in the background to let them handle events like push events. This speeds up the start of the application because it just have to be placed in the foreground. It lets the application download content before the user needs it, making the application able to change icons for certain events and making the user experience feel much faster and smoother.
The Blackberry Platform also has functionality to let an application start when the device is powered on for that always-on experience.

4.2 Connectivity

4.2.1 The problem

The whole point of the MDS is to expose an API to access resources on your device and in order for that to work a connection to the mBlox Network is needed.

4.2.2 Solution

The connection to the mBlox network can be done by using the Connector class provided by Java ME. It supports a lot of different kinds of connections, from file connections to http- and socket connections. To get a working connection using a Blackberry Device or the Blackberry simulator the developer must define how to connect to the server. There are a couple of ways this can be done and it is up to the developer to see what kind of connection is available and choose the one that suits the applications needs best. Here are some examples of connection methods:

- Connect through the Blackberry Internet Service is done by appending ”`;deviceside=false`” to the url.

- Connect directly via TCP/IP to a server is done by appending ”`;deviceside=true`” to the url.
- Connect via Wi-Fi is done by appending ”`;interface=wifi`” to the url.

For this project, as the simulator was used, the second and third, “`;deviceside=true`” and “`;interface=wifi`”, was used because the simulator cannot connect to the BIS and to connect to internet from the simulator a faked Wi-Fi connection is needed.

4.3 Registration to Mnet

4.3.1 The problem

To use the Mnet the device needs to be registered within the Mnet. The device needs a secure way to talk to the registration server to retrieve its certificate.

4.3.2 Solution

This is done using HTTPS and is handled by a special server inside the Mnet. Part of the registration process is getting a certificate from a special server inside the Mnet. The server issues a new certificate to each new device that connects and since the simulator reset on every boot the registration was mocked during testing in order not to waste too many certificates.

4.4 Security

4.4.1 The problem

The Blackberry OS has built in support for communication with TLS but it doesn't support the use of client authentication, which is a way for a server to identify and trust the client. This was a big concern because the Mnet requires the clients to authenticate themselves. An alternative to the Blackberry implemented communication was needed.

4.4.2 Solution

To solve this problem the open source library Bouncy Castle [12] was used. This made it possible to use TLS with client authentication. One key part of the security is the handshake, where the parties exchange their certificates and prove they are who they say they are. The device has a public key, certificate, from the server already stored and when the device connects to the server it matches that certificate with the certificate that the server sends. It also has a public and private key which it uses to identify itself to the server. The TLS handshake looks like the following:

- The client sends a *Client Hello* to the server.
- The server responds with a *Server Hello*.
- The server sends its *Certificate* which essentially is its public key.
- The server requests the *Client Certificate*.

- The server sends a *Server Hello Done*.
- The client responds with its *Certificate* which essentially is its public key.
- The client sends a *Client Key Exchange* with a secret that the client computes with the servers public key.
- The client sends a *Certificate Verify* with a message computed with the clients private key.
- The client sends a *Change Cipher Spec* which is an encrypted message, the server decrypts and verifies it.
- The server sends a *Change Cipher Spec* which the client decrypts and verifies.
- The handshake is now completed, all continued communication is encrypted.

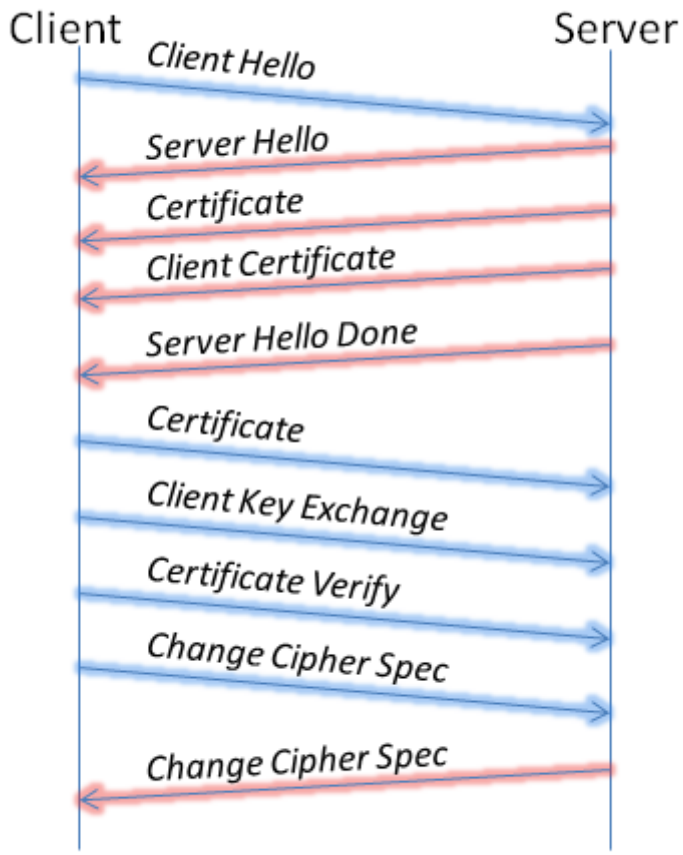

Figure 7 Chart visualizing the TLS Handshake

4.5 Blackberry Push

4.5.1 The problem

A customer might not want their devices to be connected to the Mnet at all times, maybe they have low battery at a time and closes the connection, then they would manually need to reconnect at a later time if they still want the content from the Mnet. There needs to be a way to wake the device up and tell it to connect to Mnet without the user having to do it manually.

4.5.2 Solution

This can be solved by using the Blackberry Push Service [2]. This service is offered by RIM and lets a developer send data to a device through the Blackberry Internet Service. In this case the Blackberry Push could be used to wake the device up and tell it to connect to the mBlox Network.
The Blackberry Push Service can be used either through the public BIS or through an enterprise BES. If it is used with a BES all devices connected to that BES can be addressed and if it is used with the BIS all devices registered to receive push-messages from a specific application can be addressed. To use the Blackberry Push Service through the BIS you have to sign up at the RIM website. Also to test out the Blackberry Push Service a real device must be used because a simulator cannot connect to the BIS.
When a Push Message is sent it is addressed to the devices unique Pin-number, this allows the BIS or BES to know what device to send the message to and the Push Message can contain up to 8 kb of data.

4.6 Code Signing

To control the usage of sensitive API's RIM has restricted the access to some of their API's to registered developers [3]. To register, a developer signs up to RIM and pays a small fee to get Code Signing Keys which then are used to sign any application that uses any of the restricted API's.
Some examples of restricted API's are the ones that gather information about the device, the ones that reads/writes data to the devices storage and the ones that handles encryption/decryption.
The code signing keys are not needed to test the application on the simulator.

4.7 Testing

4.7.1 The Problem

How does a developer know when an algorithm behaves right and how does the developer know that future changes does not break the functionality?

4.7.2 Solution

A developer can use automated Unit testing that runs after each change to assure that the functionality of the code is right and that it remains right after changes. This can be done on the Blackberry Platform but it does not support well known test frameworks such as JUnit or TestNG. However there are

some frameworks a developer can use that are run as an application on the device or the simulator. This means that the framework creates a GUI on the device or simulator that shows the test results. This makes it harder to have automated test cases because all tests must be run as an application on the device and as such cannot be run automatically when building the package for example.

5 Results

In the following chapter the results and the chosen solutions for the points in Chapter 4 are presented.

5.1 Background Application

As mentioned the Blackberry Platform supports applications being run in the background, also known as multitasking. This means that an application can be run in the background handling the connection to the Mnet and have one application with a Graphical User Interface, which shows the information to the user. A problem with having two applications running is that communication between the processes can be a bit tricky because they are two separate applications, this is solved by using Global Events.
Global Events are events that an application sends out on the device allowing any application listening to intercept and read the event. This makes it easy to implement communication between a background application and a foreground application. However there might be a security risk if a developer wants to send secret information, because anyone can develop an application that listens to all global events. A more secure way would be to store the information on the device, in what RIM calls the Runtime Store. That is more tricky to implement, but it would make it harder for other developers to get the information. For our purpose Global Events were used because the MDS only sends out information like “Connection Finished” and such, and that is not very secret.

5.2 Connectivity

In the early development the connection was mocked, a fake connection was implemented, to speed up development and not having to fix all steps at the same time. With a mocked connection the developer can predefine what the server should respond and therefore the development of the different stages of connection can be tested individually. When the connection felt stable enough a local Mnet was setup on the developer machine, it was configured not to use any encryption or authentication, this made testing the connection a lot easier and faster. When the connection worked properly a real developer server was used instead, it was configured to use the right kind of encryption, authentication and registration.

5.3 Registration to mBlox Network

Registration to Mnet was straight forward because the HTTPS connection is supported by default by the Blackberry Platform. It was just a matter of formatting the request right and send it to the right URL. After that the only concern is saving the certificate in a secure enough way in the device. This

was done by using a functionality called Persistent Store with Controlled Access. This allows only applications signed with the developers Code Signing Keys to access the data and makes it quite safe for normal users. Of course if a user connects the phone to a computer and hacks into the file system they can access the certificates, but even if they get a hold of the certificates the only thing they can do is to trick the Mnet that they are a device, and they will get push messages.
As mentioned before the registration was mocked during testing to prevent retrieving too many certificates from the registration server.

5.4 Security

Using the Bouncy Castle library meant that the problem with client authentication could be solved without having to implement the entire TLS protocol and handshake from scratch. Also since this is security related, implementing these features would have required some extensive testing to guarantee security. These things were estimated to be handled by the Bouncy Castle library. The documentation of the library was scarce and debugging was slowed down because the library only sent out an Internal Error instead of the real exception. The main problem with getting the communication working was to get the public and private keys in the right format for use with the library. The problem was that certificates were represented by the Certificate object and getting the data into that object was not always that easy. The same problem was that the private key to the certificate had to be represented by the class `AsymmetricKeyParameter`, but that wasn't really specified how to get the private key into that object. But after some trial and error the keys finally got into their right objects and the connection could be established. Verifying that the connection is secure enough was not done, partly because the lack of knowledge about testing secure communications and partly because we figured that if the server accepts the TLS handshake it is correct, otherwise there might be a real security issue on the server.

5.5 Blackberry Push

The Blackberry Push service was not tested in this project, mostly because to test it, a real device with a Blackberry Data Plan was needed and that wasn't available. But in theory the Blackberry Push Service could be used to send any data to the device and an application could be used to listen to those push messages and start the connection to Mnet when a message is received.

5.6 Location

One part of the Push Plug-in for MDS is that a pushed message is activated around a certain location. For example, this could mean that if a user arrives into a special town the application activates a message with special offers for

that town. This is supported by the Blackberry platform without any hassle. RIM has implemented a Proximity Listener that alerts when the device comes within the proximity of a specified location.

5.7 File access

Another plug-in for the MDS is used to manipulate files on the device, it can be pushing down a picture to the device or download a picture from the device. The Blackberry platform supports reading from the SD-Card and lets a developer manipulate some files on the device.

5.8 Testing

Most of the testing was made on the simulator without any unit testing and the testing was only done to prove that the code that was written could work. This was because of lack of automated test frameworks. Also most of the testing is only done as a proof of concept, every case is not tested and therefore there still might be bugs in the system.

5.9 Environment and Tools

Most of the development for this thesis was made on laptop running Windows Vista Business, using the Eclipse 3.6 IDE with the Blackberry Plug-in. The Blackberry 9800 Simulator was used to test the software.

6 Conclusions

Since the platforms are very much alike I was able to start working on a proof of concept solution early. Over all the result of the Proof of Concept implementation was successful, all the parts were implementable on the Blackberry platform. Here is the list from 1.3 and a short note about them.

- Secure communications – This was done by using the Bouncy Castle library because the Blackberry libraries didn't handle client authentication.
- Connectivity – This was done by using the built in libraries. They all supported socket connections and with the help of the Bouncy Castle library the connection was encrypted as well.
- Background Application – This was done using the built in support for background applications in the Blackberry platform.
- Blackberry Push – This was not done, but it is possible to send any 8 kb of data through the Blackberry Push service.

6.1 Feasibility of port

Since both platforms are quite alike the feasibility of a port from Android to Blackberry is very much possible. This made me be able to do a complete proof of concept system of the MDS.

6.1.1 Core

Porting the core from the Android platform to the Blackberry platform was quite straight forward because both the platforms use Java for their applications. They both allow true multitasking and they are designed to have applications running in the background polling for information. Connections to the Internet behave in roughly the same way, the developer opens a Socket to a server and then opens Input Streams and Output Streams via that socket.

6.1.2 User interface

Obviously one of the biggest differences of the platforms is the User Interfaces. Blackberry has only been using touch screen navigation since version 4.7 while Android was designed with touch screen navigation from the beginning, many Blackberry devices still don't use touch screens [7]. But as the MDS is a background process only meant to connect to the Mnet it doesn't have an UI and only a simple test UI was developed. The test application just showed the status of the connection and it also had the ability to show messages that were pushed to the device.

6.2 Discussion

6.2.1 Design

Even though my previous knowledge about the Blackberry platform was next to none the porting of the MDS went smooth, this has much to do with the Java platform used in both Android and Blackberry. Even though Java ME and Java SE differs in some points they are very much alike in many aspects. They both are object oriented which makes the design very much the same between the platforms. Also the syntax is the same, with the exception of some newer features in Java SE, like “for each”-statements.

6.2.2 Implemented utilities

It also turned out that Mashmobile already had used Java ME before and there were a lot of implemented utilities already, which were implemented to be like their Java SE counterparts. This made the development much easier and sped up the process of porting. Over all everything went according to plan and after just a few weeks there was a foundation to build the new Blackberry version of MDS upon.

6.2.3 Support

Since Java is one of the most used programming languages in the world if you get stuck somewhere there is probably more people out there who have been stuck on the same problem and someone else who has a nice solution. As for the Blackberry specific problems the Blackberry Support Forums [4] were quite helpful, though the developer base is not as large as on the Android platform forums, and therefore it was a little harder finding solutions to some problems than it would have been on a larger platform.

6.2.4 Documentation

Another point that slowed down development a bit was the lack of useful documentation of the Blackberry API’s. RIM has made some documentation although the explanations found there not always are very intuitive. Again, here the support forums were helpful because it’s not just me who complains that RIM has insufficient documentation in some packages.

6.2.5 Simulator

The simulator was another point of concern because sometimes it crashed with a Java Virtual Machine exception, sometimes it just did not recognize eclipse which meant difficulties to use the debugger and other times it just refused to start until the computer was rebooted.

6.2.6 Tools

To test that the MDS was working properly HTTP-requests needed to be sent to the device to see if it responded. The tool used for that was Fiddler2, it’s an easy tool to use to send custom HTTP-requests and offers an easy way to examine the response.

6.3 Further development

Further development of the MDS would be to do some proper testing, the testing that has been done so far is only to prove that it can work. Also some more thoughts about the bearer management (handling internet connectivity when changing from WiFi to 3G for example) should be done because that is also done mostly as a proof of concept. Because the MDS is only a library a proper application that uses the MDS might be a good idea to really show off the potential of the system.
To get a good library that third party developers want to use, some more of the plug-ins might need to be implemented too because the only plug that was developed was the Rich Push Plug-in that handles push-messages. Also since it is a library developed to be used by third party developers, proper documentations of the exposed API's is necessary for a third party developer to appreciate it.

6.4 Summary

The proof of concept system can be used as a library by third party developers, it has the same functionality as the MDS for Android, though it lacks most of the plug-ins and it still might have bugs in it because it is not tested very much. But the Rich Push Plug-in that is used to push out messages that are activated at a certain time is implemented and working, though this is not tested enough and might still have some bugs in it.

This has been done:

- Connectivity is working
- Security is working
- Background application is working
- A proof of concept system is working

This needs to be done:

- Proper testing
- Blackberry push needs to be implemented
- Develop a real application to show of the library

7 References

7.1.1 Source criticism

Most of these references are to the Research in Motion website. I have to assume the information on these are correct. The same goes to the Android and Mashmobile developer websites.
The Apple Push Notification Service [9] was made on a WWDC, Apple developer conference, and as such I do not have the reference to the exact announcement, just the news of the announcement. The Apple developer site does not explain any technical information about the service whereas on the conference they explained shortly how it works.

7.1.2 References

[1] http://www.blackberry.com/developers/docs/6.0.0api/index.html (May 2011) Blackberry Java API Reference
[2] http://us.blackberry.com/developers/platform/pushapi.jsp (May 2011) Blackberry Push Service website
[3] http://us.blackberry.com/developers/javaappdev/codekeys.jsp (May 2011) Blackberry Code Signing Keys website
[4] http://supportforums.blackberry.com/t5/Developer-Support-Forums/ct-p/blackberrydev (May 2011) Blackberry Support Forums
[5] Mashmobile developer documentation. This is not public. Use here is approved by Mashmobile.
[6] http://developer.android.com (May 2011) Android developer website
[7] http://us.blackberry.com/developers/choosingtargetos.jsp (May 2011) Blackberry: Choosing target device OS
[8] http://us.blackberry.com/apps-software/business/server/full/ (May 2011) Blackberry Enterprise Server website.
[9] http://www.engadget.com/2008/06/09/iphone-push-notification-service-for-devs-announced/ (May 2011) Apple Push Notification Service announced on Engadget.
[10] http://www.explainstuff.com/2009/06/17/what-is-push-email/ (May 2011) Information about Push Email .
[11] http://www.canalys.com/pr/2011/r2011013.html (June 2011) Mobile Operating System sales 2010 Q4
[12] http://www.bouncycastle.org/ (June 2011) Bouncy Castle website.

8 Terminology

- API – Application Programming Interface (Page 1)
- BES – Blackberry Enterprise Server (Page 6)
- BIS – Blackberry Internet Service (Page 6)
- IPC – Inter Process Communications (Page 9)
- Java ME – Java Micro Edition (Page 8)
- Java SE – Java Standard Edition (Page 8)
- MDS – mBlox Device Software (Page 6)
- Mnet – mBlox Network (Page 1)
- RIM – Research in Motion (Page 5)
- TLS – Transport Layer Security (Page 15)