MobiVTag
A First Development

Mattias Grafström

Université de Genève
Object Systems Group
Professor Dimitri Konstantas, Katarzyna Wac

June 2005

Table of Contents

Chapter 1 – Introduction

The MobiVTag project was presented to me by Professor Dimitri Konstantas. The idea of the project was very appealing. As it was a totally new idea, it had to start from scratch. As it is explained more in detail the MobiVTag project proposal, the idea is to use the latest technologies, in wireless communication and GPS to provide a bi-directional location based service (LBS) system for a community of visually-impaired and blind persons. The prospect of working with these technologies and serving a good cause were very motivating. An action plan for the first steps of the project was thus put in place.

This document is organized in the following way:

In chapter 2, detailed scenarios, involving visually-impaired and blind persons, are explained to demonstrate the potential use of the system.

We proceed to the requirements and design of the system in chapter 3. The functional, operational and technical requirements are defined. The design of the system is presented and both the client and server perspective are modeled and discussed.

The hardware and software configuration is described in chapter 4. The search for an appropriate mobile device is described and this appeared to be a more difficult task than planned. Further hardware and software issues are extensively detailed.

In chapter 5, we explain the building of the first prototype, which was programmed and tested. An explanation of the programming procedure and functioning of the application helps to understand how the prototype works. We also discuss the future developments needed to further advance in the MobiVTag project.

Finally in chapter 6 we summarize what we have learned in this initial study.

Welcome to the MobiVTag project!

Chapter 2 - Scenarios

The two following scenarios will give an idea of how the MobiVTag project could be used in the future.

2.1 Blind Cross-Country Skiing

Emmanuelle is 21 years old and lives in Fribourg. During her teenage, she was a promising heptathlon athlete and won several junior Swiss championships. She really enjoyed practicing her sport and was a training fanatic. She would never miss a training session unless she was nestled to her bed with fever over 39 degrees. Her dream was to compete in the Olympic Games of Athens. With her talent and motivation, she had a good chance to make her dream come true.

However in the winter of 2001, a tragic event happened in Emmanuelle's life. She lost her sight during a car accident. After a few month of depression, Emmanuelle felt she wanted to continue her sports career. She discovered the possibilities of sports for blind and visually impaired. Cross-country skiing is one of these activities. As she used to go cross-country skiing two weeks every year in the Berner Oberland to prepare herself for the summer season, Emmanuelle was familiar with the skiing technique.

Cross-country skiing for blind and visually impaired is usually performed in pairs. The blind person is accompanied by a guide, who warns about the difficulties and helps in case of problems. Emmanuelle tried it and enjoyed it very much. Nevertheless, she was dependent on the availability of the guide. She felt she would like to be able to go out on this 10km circuit on her own. As she had done the tour a number of times with a guide, Emmanuelle felt comfortable, gaining familiarity with it. As this circuit is particularly popular among persons with viewing disabilities, the MobiVTag system became very useful and has helped persons with viewing disabilities to cross-country this round without a guide. Let us illustrate this by describing a usual ski tour for Emmanuelle on a sunny day.

Claire, Emmanuelle's mother, drives Emmanuelle to the ski resort and helps Emmanuelle with her ski gear. Once she is ready on the track, Emmanuelle makes sure her mobile device is active and puts her wireless headset on her ears. She turns the MobiVTag ski application on by using the key shortcut. As she hears the welcome sound of the application, Emmanuelle is now assured everything is functioning. Emmanuelle also had made sure the battery is charged by connecting the mobile device to the power net the night before.

As it is already 13:30 in the afternoon, other blind users have taken the route and left their remarks on the MobiVTag system. Therefore Emmanuelle expects to receive some information about the condition of the track. Her mobile device makes three distinct sounds to tell her she has received three pre-race comments. A pre-race comment is a comment made by a visually-impaired skier, who has already finished the tour and wants to give his comments and warnings on the current condition of the track. Emmanuelle chooses to only listen to the last message so she presses the button on her Bluetooth headset once. The voice message is from the last blind user to have skied the track. It says that the track is calm and mostly in good condition, the temperature is pleasant but it also warns for two warning tags on the track. The first one mentioned is about a portion of track in a curve, which has been partially destroyed and where caution is needed. The second tag is about a short downhill that has become icy and can be risky. After having listened to this audio tag, the program goes back to the standby position. She presses the button on the headset once to tell the application she is starting her trip. This sends a tag with her initial position on the track, which will later let her know when she has completed the track. She puts her mobile device in the right pocket of her trousers. She is now ready to go.

Emmanuelle starts her run. It goes smoothly. The track is fine, as it was said in the audio comment. After she crosses the woods, she feels a vibration in her pocket and hears a warning sound in her headset. These are the recognizable signs of a warning tag. This means she has to be aware and slow down in the next 50 meters. Emmanuelle has the possibility to listen to the vocals by pressing the button on her headset but chooses not to. As she is approaching the curve, she knows it is the tag, which was mentioned by the pre-race comment. Being careful, she decreases her speed and feels how the condition of the track is poorer for a few moments. Once the damaged area is passed a vibration signal is created by the mobile device and Emmanuelle regains her cruising speed.

As Emmanuelle continues her route, she feels the wind is starting to blow harder and harder. She can also hear how the branches of trees are moving. Suddenly, Emmanuelle loses control of one of her skis, because of a branch that was under her left ski. She doesn't fall but as she moves on she can feel a lot of branches have fallen on the track. Therefore Emmanuelle decides to create a warning tag. She pushes the button on her headset during two seconds and the application understands Emmanuelle wants to create a tag, but still doesn't know what kind of tag. The application says: "What type of tag would you like to create?", Emmanuelle answers: "Warning". The voice recognition module translates Emmanuelle's voice into a command inside the application. As the application has been preconfigured to understand Emmanuelle's commands. The application continues with the collection of the information and asks: "What is the content of your warning tag?". Emmanuelle gives her comment: "For a period of 50 meters, the track has a poorer condition because of branches on the track, please watch out". Emmanuelle presses the headset button to tell she has finished recording her message. The application confirms the reception of the message. However this information is not enough for the program to create an appropriate tag. The program has a precise position of Emmanuelle, with help of the GPS integrated to the mobile device and considering the fact she is an open field, without buildings shadowing her from positioning satellites. Yet the defective area can't be defined accurately without Emmanuelle's help. She needs to tell the application when the tag starts. Hence the program asks Emmanuelle how far back on the track she wants the tag to start so that the skiers are warned in due time. She gives an approximate metric (5, 10, 15, etc) reply through her voice, which is again recognized by the system. The system then calculates the area of the tag, depending on the direction of Emmanuelle. As this is not a permanent problem, the tag availability will be given the standard value and will expire at the end of the day. The tag has now been created and is then made available to the other users.

Of course all the skiers have different speeds and Emmanuelle as a top athlete is a fast skier. So she catches up with the skiers who are in front of her. This is what is happening now. The skier in front of her is also visually impaired and is thus also equipped with a mobile device using the MobiVTag system. As the server of the system receives the position of the skiers nearly instantly, it can calculate their actual speed and therefore warn them about other skiers getting closer. Emmanuelle receives a collision warning in her headset and feels specific vibration in her pocket as the skier in front of her only is only 20 meters away. She thus changes track to overtake him. Her counterpart skier was also warned but with the other type of collision warning, as it is a skier coming from behind.

Emmanuelle receives the second warning tag, mentioned in the pre-race comment. She chooses to listen to it by pressing the button on her headset. It tells her the details about the icy slope in the following downhill. So she takes it easy and passes the obstacle without problem. This was the last major hurdle of her tour. She has about 10 minutes left on the tour so she calls her mother to tell her to pick her up by using the voice dialing feature on her mobile device and the call button on her headset. An end of tour signal is sent to her mobile device, when she passes the point from where she started and sent her initial position. Her tour is finished. She can decide to leave a pre-tour comment, which can be used by the fellow users. It is the same type of message she listened to before she started her tour.

Emmanuelle's tour has enabled us to see some of the different features applicable to the MobiVTag system. There are other features, which didn't come out through her trip.

2.2 Visually-Impaired in Town

Jean is 25 years old and lives in Lausanne. He has had serious sight problems since he was born. At the moment, he is studying economics at the University of Geneva. Every week day, he takes the train to Geneva and the tram from the station to the university. Jean is always accompanied by his guide dog Alfie. Jean has been a user of the MobiVTag system for a few months now and we will illustrate its usage by following him during a normal school day.

Jean wakes up and prepares himself to go to university. He takes his mobile device (an advanced 3G phone with the client application of MobiVTag installed on it), that has been charging the batteries during the night and turns it on. A Bluetooth headset connected to the phone is also part of his equipment. Jean and Alfie are now ready to go. This is a route, which Jean has taken over a hundred times but new things, changes or temporary disturbances can always appear due to the dynamics of a city. As Alfie is leading Jean to the station, Jean comes across a car which is partially parked on the pavement. As other blind or visually-impaired might take this path, Jean thinks it would be appropriate to create a tag. Users with sticks would find a vehicle on the pavement a very disturbing obstacle. Thus he takes his mobile device and enters the key code for a minor obstacle. A minor obstacle is one of the templates used in the MobiVTag system. As a tag has a lot of different values, it would take a lot of time for a user to enter them all. Therefore templates tags, with standard values for the tag, have been created for the type of occurrences, which appear frequently. After Jean has pressed the key code, the mobile device sends the values of the tag to the server. This includes the position, where Jean is standing, the area of validity of the tag, an area of 10 meters around the car, the time of validity of the tag, unlimited in this case, until someone cancels the tag when a user observes the car has left the pavement. All this information is generated by the template or automatically by the mobile device, to leave the user with as little work to do as possible. Alfie and Jean continue their walk to the station.

Once they have arrived to the station, Jean and Alfie enter the train in the direction of Geneva. They arrive outside the railway station of Cornavin. Once outside, Jean receives a coded message on his mobile device. The phone vibrates twice. This is the signal for an important tag, which needs audio reception. Thus Jean reaches to his Bluetooth headset and presses the audio button to listen to the audio recording of this tag. This tells him, that there are major road constructions around the station as the station square is being renovated. An alternative itinerary is proposed to get to the different areas around the station. This is a large area tag, which is sent to users around a large area around the station and will be valid until the construction work is finished. Taking the information in consideration, Jean and Alfie follow this new route, with Alfie helping Jean to avoid some pole lights.

They arrive to the tram stop and take the tram to the university. Jean goes to his courses. Today he finishes around 2 in the afternoon. It leaves him time to meet one of his friends in a café of the old town, where they usually meet. On the way to the café, Jean receives a signal on his phone, it vibrates once. This is the code for a minor obstacle. However as they walk in this area, there is clearly no obstacle. The obstacle has obviously been moved since the person that created the tag passed in this area. Thanks to his mobile device, Jean can update the non validity of the tag by pressing the delete tag key code on his phone. The system is by the users, creation, amending and deletion of tags are user-based.

As Jean walks up the hill, he can hear noises of construction works and Alfie leads him to the other side of the road. Jean needs to create a tag, as this area was not mentioned by a tag. However Jean doesn't know the size of the area, he leaves this field open in his tag and asks the system to flag this tag as incomplete. According to the direction in which Jean was walking, the system can create a temporary target area. If someone comes from the other direction, the user will be asked to precise the area of the tag. The tags precision will then be more accurate and will complete Jean's information.

Jean arrives to the café and meets his friend. After spending some time with his friend, it's time for Jean to go home. He heads with Alfie to the station. On the road to the station, he receives a warning of a minor obstacle. He receives the warning a few meters before the obstacle. As he is a fast walker, the warning is sent to him a bit earlier than to other users to leave the same amount of time before an obstacle to each user. This is possible via the system, which calculates the average speed of the users and warns the users in due time, according to the walking speed. Jean avoids the obstacle, as Alfie helps him to find his way. Jean presses the confirm button on his headset to confirm that the tag is still valid.

Jean continues his trip to the station to take his train back to Lausanne and get home. Jean's tour has provided us with some ideas of how the MobiVTag can be used during an everyday-life situation of a visually-impaired person.

Chapter 3 – Requirements and Design of the System

3.1 System Requirements

3.1.1 Project Presentation

This project is an integrating part of the MobiVTag project. Details about the MobiVTag project can be found in the MobiVTag project proposal. The idea is to provide a basic infrastructure including software and hardware, which can be used to further implement the MobiVTag project.

With the help of a mobile phone, which has a GPS receiver integrated, a MobiVTag user will be able to receive virtual tags containing information about his actual position in space. The system sends a tag to the user each time there is a relevant tag available corresponding to the user's actual position.

Moreover a user will also be able to create tags with information about his actual position. The system is thus community-friendly as it is the users, which create the information available in the system. The system is totally dependent of the users input.

3.1.2 Functional and Operational Requirements

The tag is the key notion in the MobiVTag system. The requirements are associated to this notion.

? Tag Creation: A user has his mobile device and is walking in the street. He arrives at a particular place, which he wants to tag. He takes his mobile device and enters the tag parameters on his mobile device. The tag is created.

? Tag Reception: A user is walking in the street and arrives in a zone, which was tagged by a user who previously wandered in this area. The system sends the tag to the user and it arrives on the user's mobile device for him to read.

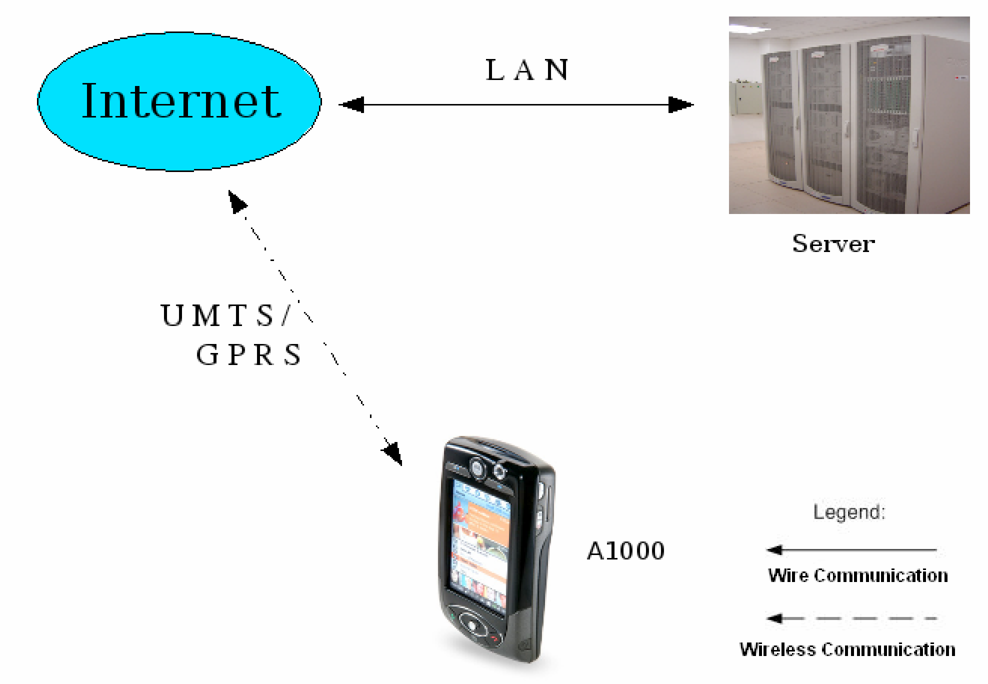
The tag concept is further explained in the tag format specification.

3.1.3 Technical Requirements

The system that will be created is a mobile system, using advanced communication and programming technologies. The system will use the client/server model. Both 2.5G and 3G mobile technologies are intended to be used to communicate. When available 3G technology will be used primarily as the performances obtained by this technology are far more superior. Thus the mobile devices will use UMTS (3G) when possible and if needed GPRS (2.5G) to connect to Internet and communicate with the server.

However as the 3G mobile phones available for the moment in Switzerland are very limited, the phone had to be imported from Sweden. The phone that was chosen is the Motorola A1000. It works on Symbian OS and can run application programmed in Java 2 Micro Edition (J2ME). Thus the client application on the mobile phones is written in J2ME. It has both a GPS included and UMTS technology. The choice of the mobile phone will be more in detail explained in section 4.1.

The server will be connected continuously to the Internet through a LAN connection to listen to the different requests coming from the client applications. It will be programmed in Java 2 Standard Edition. The server will connect to a database system, in which the tags are stored, using JDBC. The tags will be formatted using XML.



Hardware Architecture with Motorola A1000

3.2 System Design

3.2.1 Actors interacting with the Systems

Let us look at the system as a whole entity or as a black box. What happens inside the system will be looked into later. Who and what will be interacting with the MobiVTag system? As we have been able to see in the definition of requirements, the key actors are the so-called "mobile users", who interact with the system. With the help of their mobile phone, they can create tags in the system and should receive tags from the system according to their position.

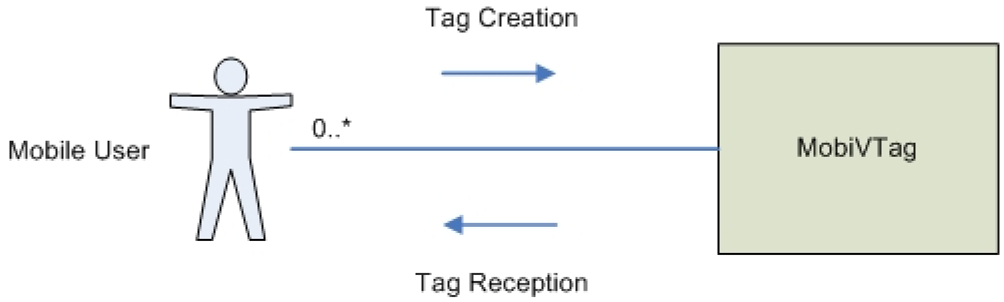
3.2.2 Identifying the Messages

The messages sent in between the systems and the actors are as follows:

- The MobiVTag system sends a message, shaped as a tag, to a mobile user when this mobile user reaches a position, which corresponds to a tag[1].
- The MobiVTag system receives a message from a mobile user when this mobile user has decided to create a tag by using his mobile device.

3.2.3 Dynamic Collaboration Diagram

From the information collected above we can create the corresponding dynamic collaboration diagram:

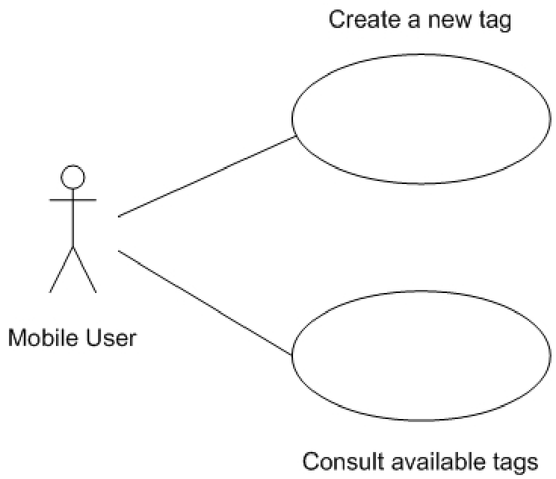


[1] To implement this, requests are sent on a regular basis from the client application to the server application of the system without the interaction of the user. This will be explained more in detail in section 3.4.

The number of mobile users interacting with the system can vary from 0 to an undefined limit.

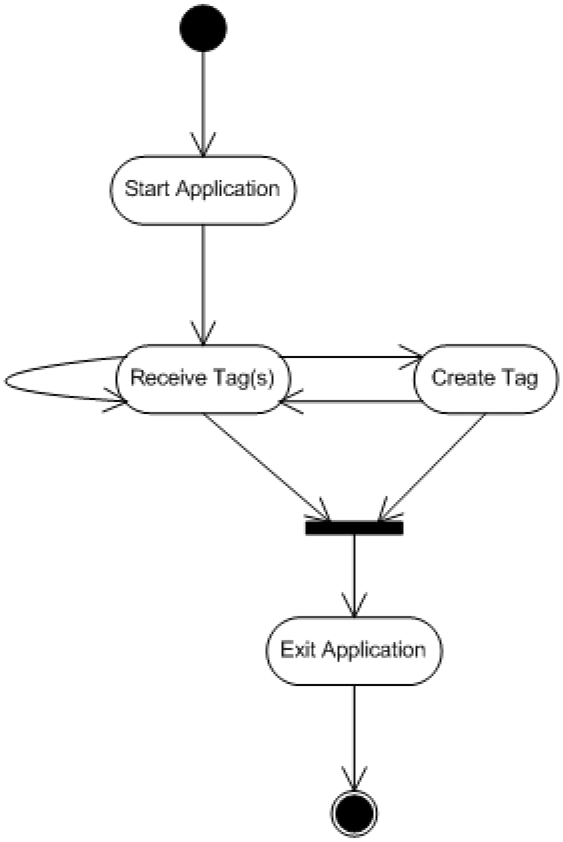
3.2.4 Use Case Diagram

From the user point of view, this gives the following use case diagram:



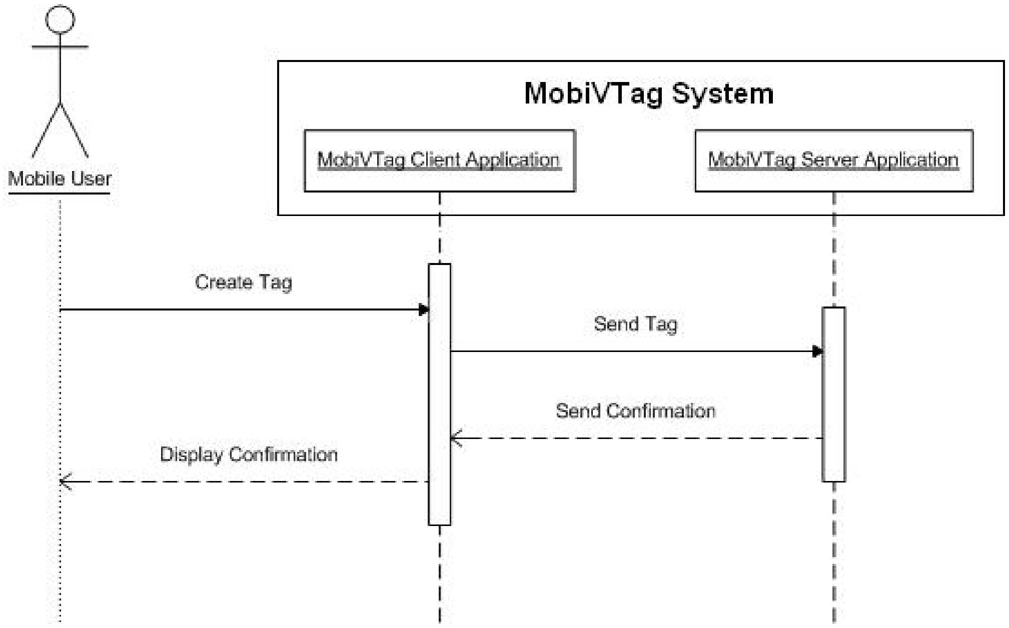
3.2.5 Activity Diagram

Accordingly, this activity diagram presents the lifecycle of the client application:

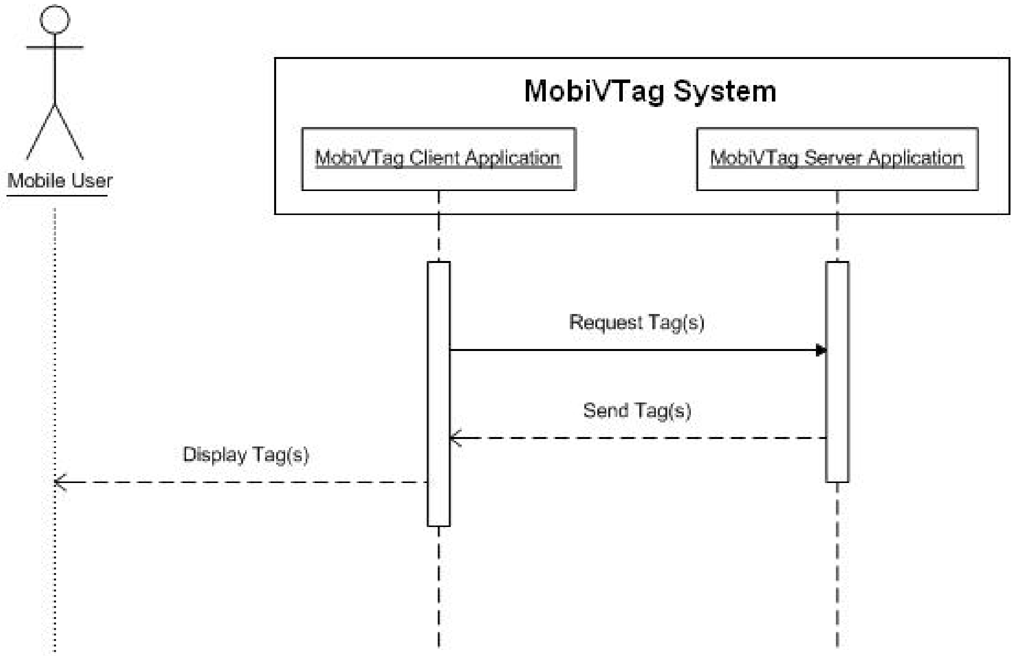


3.2.6 Sequence Diagrams

The following two sequence diagrams show the interactions between the user and the system’s two components, the client application and the server application. The first one presents the tag creation procedure:



The second one presents the tag request procedure:



3.3 Tag Format

The tag is the key concept in this system. Therefore it is essential to look at it more in detail, to enable modeling the system. The format of the tag expresses which parameters will be included in a tag. This helps us to see what kind of information is needed for a tag. From there, we can start to look at where we should find this information and which communication channels are to be used. It is the starting point for the building of the system.

The table describing the tag format and its different areas can be found in Appendix 1.

3.4 Client Perspective

3.4.1 Introduction - What is the program on the mobile device supposed to do?

? Communicate with GPS receiver to receive location information

? Send location information about the position regularly to server to search for possible tags

? Receive tags from server based on location

? Create tag with location information when requested by the user

? Send tag to server

? Have a simple graphical user interface

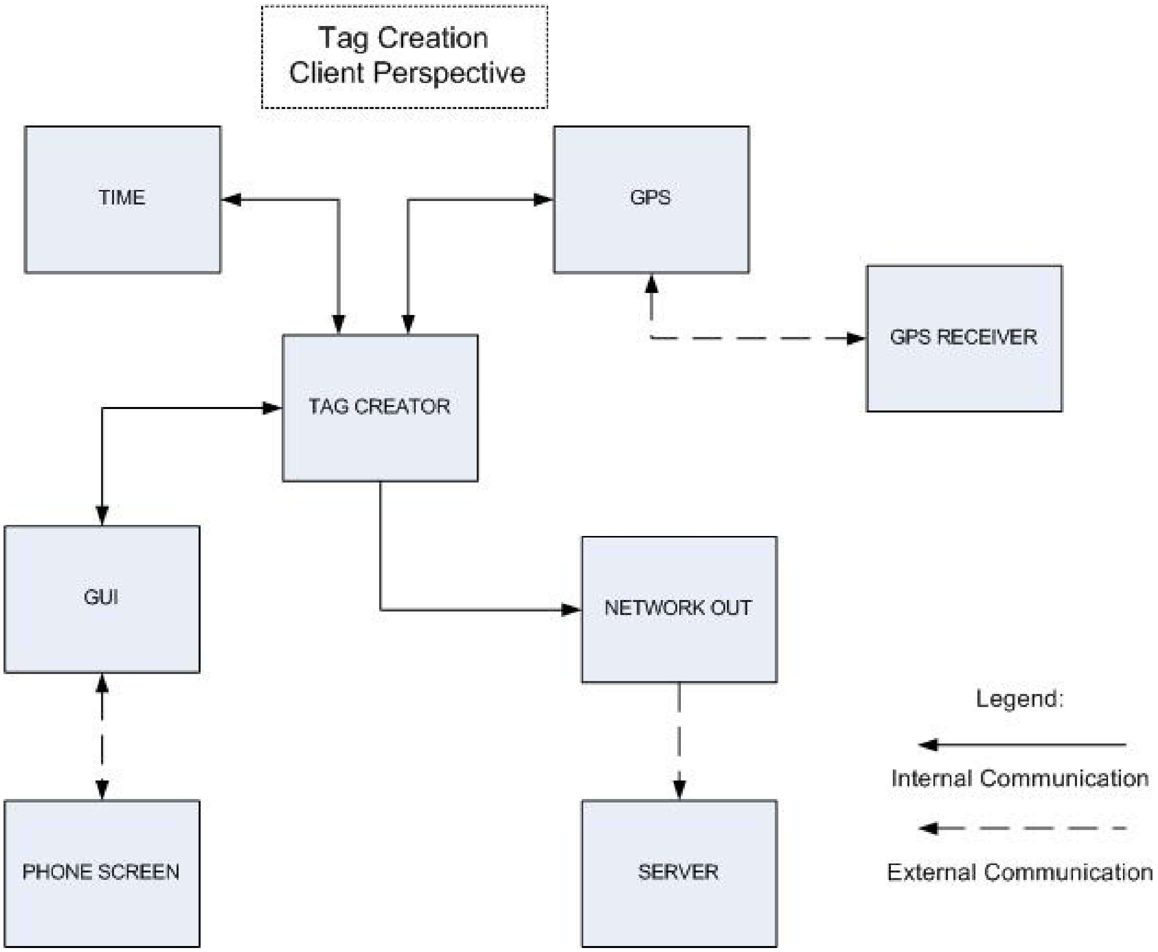
3.4.2 Creation of a Tag

3.4.2.1 Chain of Events

1. User presses button "create tag"
2. Program searches for location information through GPS receiver
3. Tag is created according to the users input
4. Communication channel is created to the server
5. Tag is sent to server
6. Confirmation of reception is received
7. Communication channel to the server is closed

3.4.2.2 Functional Blocks Diagram

This diagram shows the functions, which will be used in the tag creation process and the communication flow internal and external of the client application.



3.4.2.3 Description of the functionalities used in the tag creation process

? GPS Function

- o Requests and receives location information from GPS RECEIVER
- o Sends location information to TAG CREATOR and TAG REQUESTER

? GUI Function

- o Interacts with user through MOBILE PHONE SCREEN
- o Sends information to TAG CREATOR when user has asked to create a tag

o Receives information from TAG RECEIVER when tag is received and publishes it

? NETWORK OUT Function

o Sends tags and tag interrogation requests to SERVER

o Discovers the services available and communicates with the SERVER

? TAG CREATOR Function

o Communicates with GUI and receive user input

o Requests and receives location information from GPS

o Requests and receives time information from TIME

o Creates tag

o Sends tag to NETWORK OUT

? TIME Function

o Requests and receives date and time information from SYSTEM

o Sends date and time information to TAG CREATOR

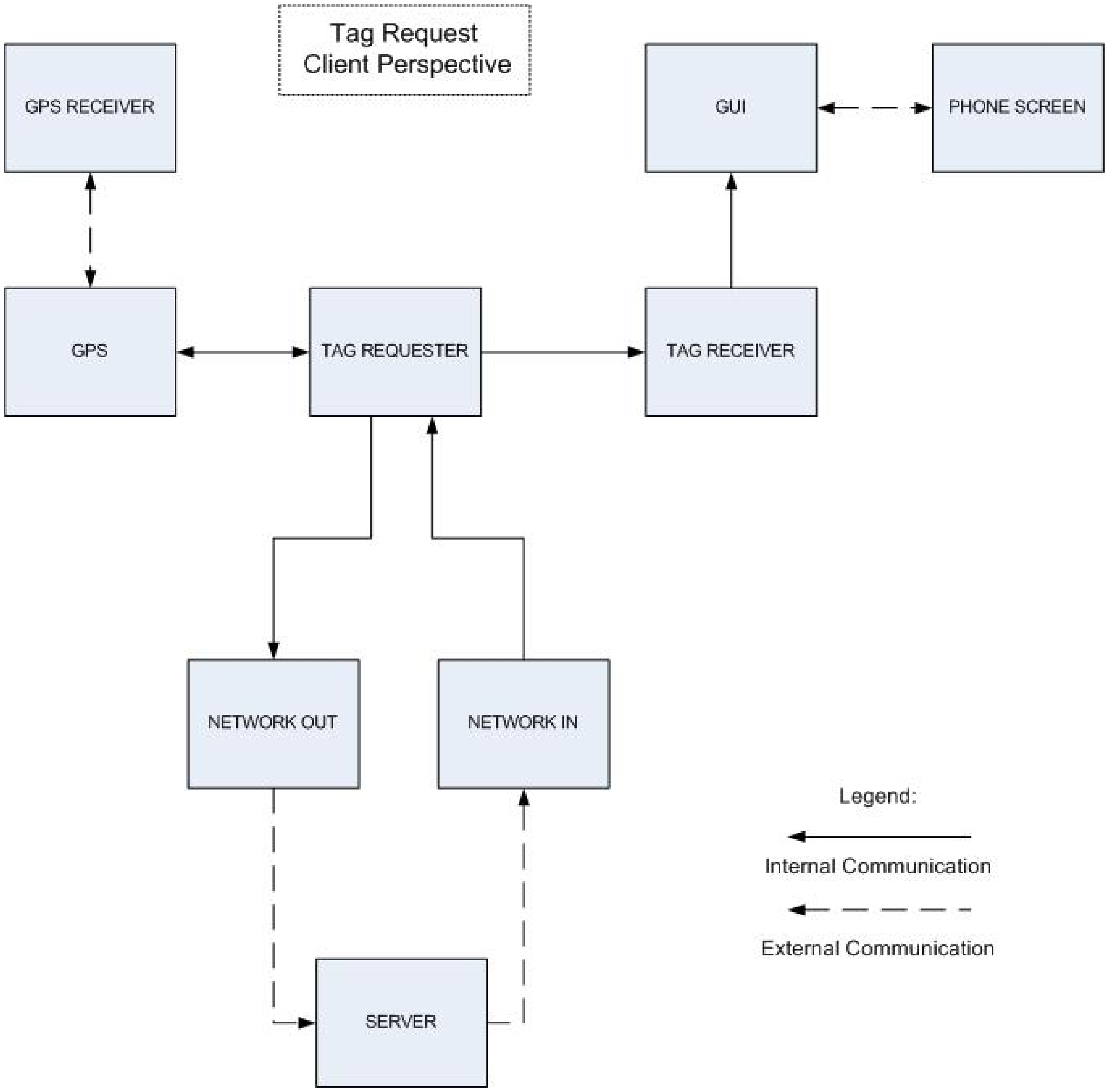
3.4.3 Request of a Tag

3.4.3.1 Chain of Events

1. As the user is moving, the client application detects a significant position change while questioning the GPS receiver
2. A tag request is created with the new location information
3. A communication channel is created to the server
4. The client application sends the tag request
5. The server sends a reply with either a tag or a negative answer
6. Communication channel to the server is closed
7. If a tag has been received, the client application sends a message to the user via the mobile device
8. The user chooses to view the tag
9. The tag is presented to the user

3.4.3.2 Functional Blocks Diagram

This diagram shows the functions, which will be used in the tag request process and the communication flow internal and external of the client application



3.4.3.3 Description of the functionalities used in the tag request process

? GPS Function

o Requests and receives location information from GPS

RECEIVER

o Sends location information to TAG CREATOR and TAG REQUESTER

? GUI Function

o Interacts with user through MOBILE PHONE SCREEN

o Sends information to TAG CREATOR when user has asked to create a tag

o Receives information from TAG RECEIVER when tag is received and publishes it

? NETWORK IN Function

o Receives results on tag interrogation request from SERVER

o Sends results on tag interrogation request to TAG REQUESTER

o Receives tags from SERVER

o Sends tags to TAG REQUESTER

? NETWORK OUT Function

o Sends tags and tag interrogation requests to SERVER

o Discovers the services available and communicates with the SERVER

? TAG RECEIVER Function

o Receives tags from TAG REQUESTER

o Communicates with GUI

o Sends tag information to GUI

? TAG REQUESTER Function

o Requests and receives location information from GPS

o Determinates when to make a tag interrogation request depending on position change or time

o Creates tag interrogation request

o Sends tag interrogation request to NETWORK OUT

o Waits for answer on interrogation request from NETWORK IN

o If a tag is available it sends the tag to TAG RECEIVER

3.5 Server Perspective

3.5.1 Chain of Events

3.5.1.1 Tag Creation

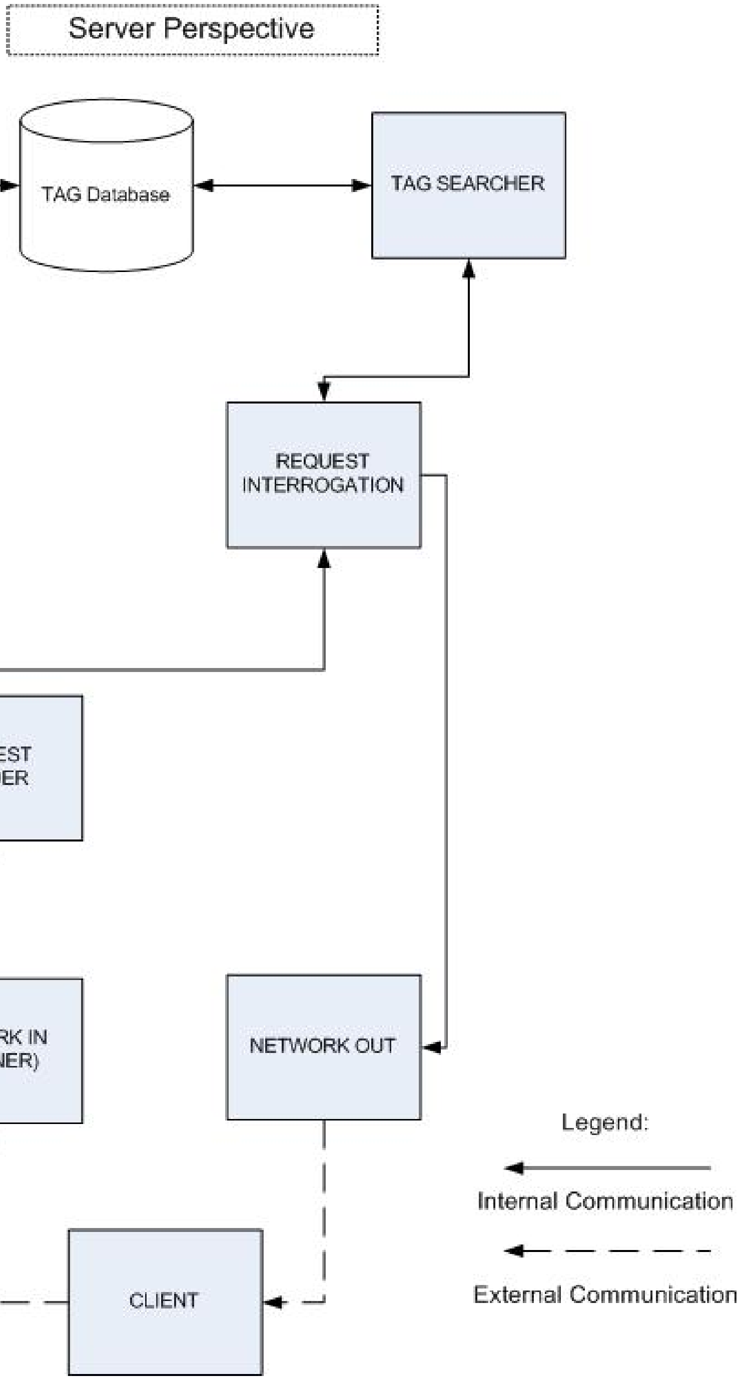
1. The server is listening to the network
2. The server receives a request
3. A communication channel with the client is created
4. The request is received
5. The server application determines that this is a tag creation request
6. The server sends the tag information to the tag database where the tag is created
7. The servers sends a tag creation confirmation to the client
8. The communication channel with the client is closed

3.5.1.2 Tag Request

1. The server is listening to the network
2. The server receives a request
3. A communication channel with the client is created
4. The request is received
5. The server application determines that this is a tag interrogation request
6. The server interrogates the tag database to see if a tag is available for the location sent by the client
7. The server sends the tags available for the client's position or a negative reply
8. The communication channel with the client is closed

3.5.2 Functional Blocks Diagram

This diagram shows the functions, which will be used in the tag creation process or the tag interrogation request and the communication flow internal and external of the server application



3.5.3 Description of the functionalities used in the server application

? NETWORK IN (Listener) Function

- o Listens to the network for tag interrogation requests or tag installations
- o Forwards tag interrogation requests and tag installations to

REQUEST DECIDER

o Discovers the clients and their availability and communicates with CLIENT

? NETWORK OUT Function

o Receives tag installation confirmation from TAG INSTALLOR

o Sends tag installation confirmation to CLIENT

o Receives tag interrogation request reply from TAG INTERROGATION

o Forwards tag interrogation request reply and if available tag to CLIENT

? REQUEST DECIDER Function

o Receives tag interrogation requests and tag installation from NETWORK IN

o Forwards tag installations to TAG INSTALLOR

o Forwards tag interrogation requests to TAG INTERROGATION

o Implements a security policy

? TAG INSTALLOR Function

o Receives tag installation request from REQUEST DECIDER

o Creates tag in TAG DB

o Sends tag installation confirmation to NETWORK OUT

? TAG INTERROGATION Function

o Receives tag interrogation request from REQUEST DECIDER

o Forwards tag interrogation request to TAG SEARCHER

o Receives reply from TAG SEARCHER

o Sends reply and if available tag to NETWORK OUT

? TAG SEARCHER Function

o Receives tag interrogation request from TAG INTERROGATION

o Searches Tag DB according to TAG INTERROGATION

o Sends reply and if available tag to TAG INTERROGATION

Chapter 4 – Hardware and Software Configuration

4.1 Choosing a Mobile Device

The first task was to find a mobile device, which suited the needs for the project. In recent years a jungle of mobile phones and PDA's has come out on the market and thus it was a necessity to define our needs very precisely. We quickly found out that the three key features for our project are:

? Connectivity: Ability to connect to the internet, through at least GPRS and preferably UMTS

? Programmability: This device should be programmable

? GPS: This device should have a GPS integrated or should have the possibility to have an external GPS connected to it

We decided to look in a worldwide context, and investigate all the known mobile phone brands. We used the internet to explore the over 30 mobile phone brands and found everything from regular mobile phones, smart phones, and advanced PDA's. However only very few had an integrated GPS module. As 3G telephones are in early development, only innovative brands had those devices in their product range.

In Appendix 2, 3 and 4, you can find comparative charts, between the different phones, which were considered. The phone that attracted our attention was the Motorola A1000. It is a 3G phone, able to connect on the GPRS and UMTS networks. As it uses Symbian OS, it is both programmable in Java 2 Micro Edition and C++. It also has a GPS module integrated. It became a clear choice, as this device really offers all the aspects we were looking for.

A backup plan was also created in case the A1000 was impossible to buy. A Sony Ericsson P910, also using Symbian OS and programmable in Java 2 Micro Edition, was considered, adding an external GPS module connecting through Bluetooth. As the later device isn't a 3G telephone, it wasn't appropriate for our project and would only be used if no other solutions could be found. Subsequently, we concentrated our efforts on the acquisition of the Motorola A1000.

Motorola A1000

4.2 Buying the Motorola A1000

Motorola may be a worldwide company. It doesn't mean it is possible to buy all of their mobile phones at any places of the world. The Motorola A1000 doesn't exist in Switzerland yet, although Switzerland already has a functioning UMTS network. Motorola with Swisscom choose for a start only to commercialize a relatively old and not particularly advanced device on the Swiss market. However the A1000 is available, where 3, an international 3G mobile network operator, is present, thus in Sweden, Italy, Austria and the United Kingdom.

An e-mail conversation with Motorola Switzerland suggested the phone would arrive in February on the Swiss market. We are in June now and the A1000 has made its appearance on the Swiss Motorola website but the phone is still not available in any Swiss shops. We made the good decision not waiting for the Swiss launch of the phone and in stead investigated how to buy the phone abroad.

E-mail conversations with 3 in Sweden suggested it would not be possible to buy the phone unless the person buying the phone is officially living in Sweden. As we have relations in Sweden, we contacted a trusted person and asked him to buy the phone for us. A money transfer was made to a Swedish bank and the phone was bought in a shop in Uppsala. The phone was bought for use with a prepaid card, without a subscription. The buy of a welcome package was mandatory. Although there wasn't any subscription involved, the phone was still blocked to the 3 network. This means that if you don't put a SIM card from 3, the phone will not work. We decided to send the phone to a company, specialized in unblocking phones. The person in Sweden took care of this and the phone came back five days later. The unblocked A1000 was then sent by Swedish post to Switzerland and received in the mailbox three days later, having survived the trip across Europe.

4.3 Connectivity of the Motorola A1000

An Orange SIM card was put in the phone to test it and the phone was working with this card. It had thus been successfully unblocked. The next step was to subscribe to a Swiss mobile provider. As Swisscom at that time was the only operator offering an access to the UMTS network, the choice was not too difficult. The only choice remaining was the type of subscription. A short market study gave us the answer. Swisscom offer a data only subscription called "Natel Data Basic", enabling connections through all the different networks, including GPRS and UMTS. As there are no particular needs for phoning with the mobile device, this offer was chosen and a subscription was signed.

However this didn't mean that the phone would connect instantly to the internet. The GPRS settings of the phone were still set to the 3 network. Automatically the phone tried to connect using the default 3 network settings. It was stated nowhere how to change them. They appeared to be blocked to 3. Luckily a free application called DefaultGPRSEditor was found on the web[2], giving the possibility to change these settings. The settings for the Swisscom GPRS network were found on their website and entered into the DefaultGPRSEditor application on the A1000.

Finally, the A1000 was thus able to connect to the internet in Switzerland and this against the wishes of the 3 network and Motorola. It

[2] DefaultGPRSEditor is available at http://nop.at/dge/

uses the UMTS network when available and switches to the GPRS network in other cases.

4.4 Installing Software on the Motorola A1000

To install applications on the Motorola A1000, we had to first install the Motorola Desktop Suite available on Motorola's developer website www.motocoder.com. After pairing the A1000 with a regular PC, we used the Bluetooth connection to connect the two. The Desktop Suite then enables the installation of applications on the A1000 from the PC.

4.5 Configuration of the Server

We chose to use an Apache Tomcat 4.1 Server, which was installed on a home PC running Windows XP. The server was configured to listen to port 8071. The server is connected to a TV-cable internet connection, passing through a router. The router is configured so that requests on port 8071 are forwarded to the PC hosting the server. The PC is online and connected permanently.

4.6 IDE Environment

jEdit 4.2, a free java editor, was used when coding. The J2ME Wireless toolkit, provided by Sun Microsystems, was used to compile, debug and package the client application, before installing it on the A1000 through the Motorola Desktop Suite. The Motorola SDK provided on the motocoder website was not used as it didn't offer access to the needed API's. This will be explained more in detail in section 5.3**.**

4.7 Use of Blog

A blog, powered by WordPress, was created to provide details of the advancement of the project. The key documents were published there. Comments could be made directly on the site to enhance communication around the project. The blog was revamped after the arrival of the mobile phone with the latest WordPress version. The address of the blog is: http://mobile.esquive.org

Chapter 5 – Programming a Prototype

5.1 First steps in Java 2 Micro Edition

5.1.1 Understanding Java 2 Micro Edition structures

Having never programmed with Java 2 Micro Edition (J2ME), it was a challenge to get acquainted with this technology. The first task was thus to read as much documentation as possible about it and get familiar with the basics. Of course as it is part of the Java family and as I already was familiar with Java 2 Standard Edition (J2SE), it was not totally unknown territory. Still it was still important to have a full overview of this specific language to understand how to use it and what its capabilities are.

J2ME is in fact the smaller brother of J2SE and the server-based J2EE. The main difference is that they target different types of devices. J2ME provides a development environment for a range of small, constrained and mobile devices. These devices can be characterized by limited capabilities, i.e. smaller computing and memory capacities. They include for example mobile phones and PDA's. There are many similarities though between J2ME and J2SE, as J2ME has been derived from J2SE. It thus also shows all the characteristics of the Java language (portability, use of libraries, etc)

There are different types of devices, which have limited capabilities and thus J2ME has been divided in two configurations:

- Connected Device Configuration (CDC) for devices with more memory, faster processors and greater network bandwidth, as for example navigation and telemetry systems.

- Connected Limited Device Configuration (CLDC) for devices with intermittent network connections, small processors and limited memory, as for example pagers, mobile phones, PDA's and smart phones.

Moreover, J2ME has been divided in different profiles. A pager clearly doesn't offer the same possibilities as a mobile phone, though both have limited network connectivity and memory. This is where the profiles are relevant. Whereas CDC and CLDC provide the lowest common denominator for a group of devices, profiles add a layer on top of the configuration providing API's for a specific class of device. One of them is the Mobile Information Device Profile (MIDP). MIDP offers the core functionality required by mobile application, such as the user interface, network connectivity, local data storage and notably, application lifecycle management.

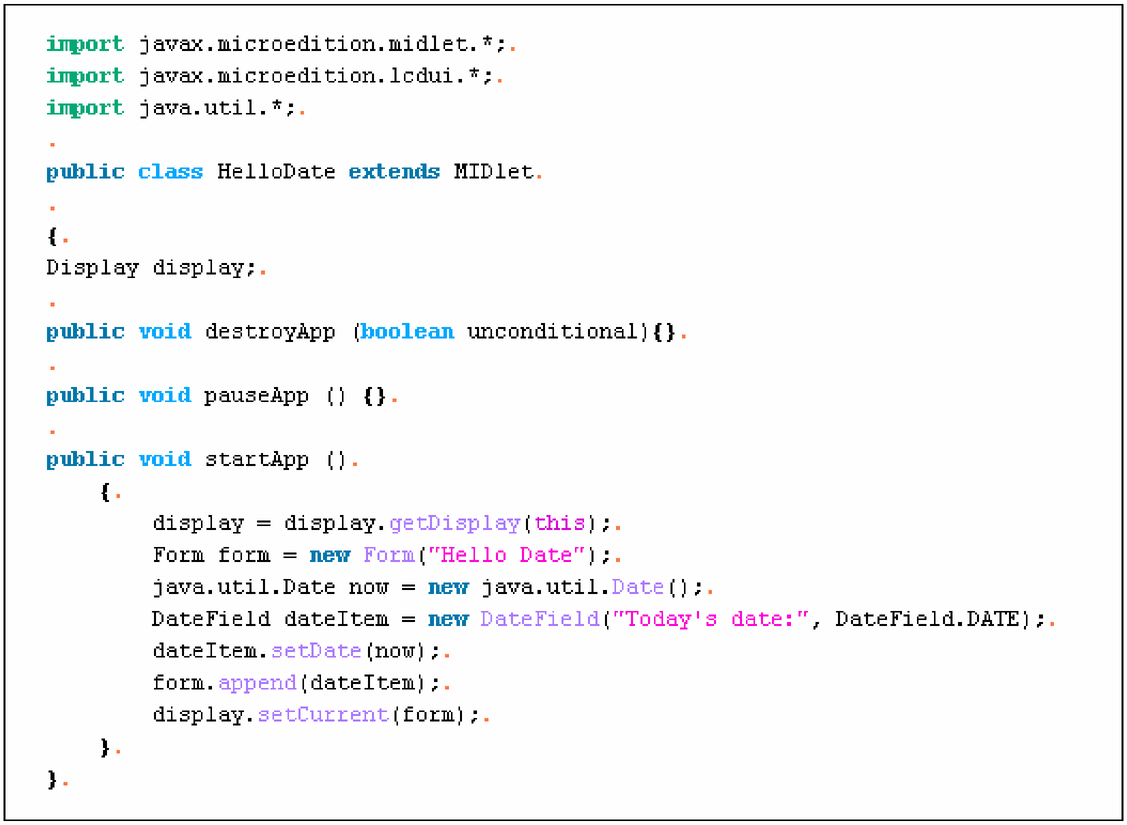
It became quite clear in which direction we were to continue. The mobile phones that exist at the moment only enable the use of the CLDC configuration with help of the MIDP profile. We were therefore bound to this technology. Rightly so as MIDP provides us with the tools we need to create our desired application. In order to understand the subtleties of MIDP programming, we started by creating a first application called "HelloDate", which we will describe in the next section.

5.1.2 A first application – HelloDate

The idea with this application was to create a simple program that shows the date on the screen of the mobile phone. It also helped testing the IDE, the connection between PC and mobile phone, as well as the installation of a newly created midlet.

A midlet is a J2ME application that executes on CLDC under MIDP. All java applications on current mobile phones are midlets and follow therefore specific rules. A midlet application must extend the MIDlet class, which can be found in the `javax.microedition.midlet` package. The application is managed by the Application Manager Software (AMS). The AMS is a part of the device's operating environment and guides the midlet through its various states during the execution process. Unlike a J2SE program, midlets don't have a `public static void main()` method. A midlet follows a certain lifecycle. It can either be "paused", releasing shared resources and becoming passive, "active", acquiring required resources and setting the current display, or "destroyed", releasing all resources. One method for each state (pauseApp(), `startApp(), destroyApp()`) has been defined and can be called by the AMS to indicate to the midlet that it should enter that certain state. When writing the HelloDate midlet, we thus had to use the structure presented above. The code, that was written to present the date, was as a result put in the `startApp()` method and was run when the application was started. The second key feature in J2ME, which was used creating this program, was the user interface (UI) in the LCDUI model. A `Form` object was used to enter the text on the screen. The `display` class, which acts as the display manager for the midlet, was used to set the display to the Form with the Display.setCurrent(Displayable) method.

The creation of this basic application helped getting an overview of how to use J2ME. The next phase was to start focusing on the prototype of the MobiVTag project and as a first step we created a simple midlet using a network connection between the phone and the server. The sample code of the application is available below.



```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

public class HelloDate extends MIDlet

{
Display display;

public void destroyApp (boolean unconditional){}

public void pauseApp () {}

public void startApp ()
    {
        display = display.getDisplay(this);
        Form form = new Form("Hello Date");
        java.util.Date now = new java.util.Date();
        DateField dateItem = new DateField("Today's date:", DateField.DATE);
        dateItem.setDate(now);
        form.append(dateItem);
        display.setCurrent(form);
    }
}
```

5.2 Programming the prototype

5.2.1 Design Pattern: Model-View

We have chosen to use the model-view design pattern[3] for the client application of our prototype. It is a simplified version of the Model-View-Controller design pattern. The classes of the program are divided in two component groups:

- Model, which manages the application's data. It responds to queries from the views regarding its state when requested to do so by the views. It also notifies the views when the state of the data has changed
- View, which presents a view of the model data. It responds to user input, instructing the model to update its data accordingly. On notification of changes to the model data, it retrieves the new model state and renders a view of the latest state of the data.

It is essential to separate the UI from the core application login. This architecture enables portability and ensures flexibility in the programming. We will study more closely the class architecture and the division of the classes in these two component groups in section 5.2.3 and 5.2.4.

Furthermore, as seen in Chapter 3, the system will also follow the classic client – server architecture. We will have one server, listening for requests from clients and responding accordingly.

5.2.2 First Development Phase: Connectivity

The idea with this first prototype was to create a simple communication link between a client and the server. On the screen of the mobile device, the user could enter a few characters in two different text fields, name and position. The characters entered in each field were then passed as parameters at the end of the URL of the server. This URL was used in a GET request method, which was the method employed to send the data to the server.

The server was constructed as a servlet, listening for get requests. It read the parameters and sent a reply with a confirmation message and

[3] Available online at: http://doc.trolltech.com/4.0/model-view-programming.html Trolltech, 2005.

the content of the parameters. The reply of the server was then read by the client and shown on the screen of the phone.

This ensured that the communication infrastructures were functioning well. The GPRS and UMTS network available on the phone were responsive and the server was reachable through the internet.

5.2.3 Second Development Phase: Tag Creation

5.2.3.1 Client Perspective

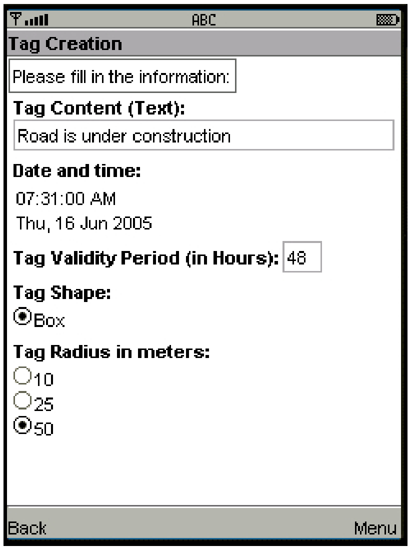
Creating a tag is one of the essential parts of the application. The user should have the possibility to create tags with his mobile device. As we had defined the tag format and its different parameters earlier, we had to visualize how this tag creation procedure would be on the prototype. As it is difficult to implement all the ideas around the project directly, we subsequently chose to limit the choices of the user.

The tags created with our prototype will only convey text. Some of the parameters will be given default values and be invisible to the user. The user will be able to:

- Enter a text, which will become the content of the tag.
- Choose a date and time of creation of the tag. The actual date and time of the system are the default value.
- Enter a certain amount of hours of validity, until the tag expires.
- Choose an area of validity of the tag, around the actual position. The area is defined as a quadratic box around the position. The size of the half side of the box can be chosen as 10, 25 and 50 meters.
- Note that the shape of the tag is set to box shape.

The box shape was chosen for convenience. Calculating the relevant positions for the case of a circular tag area would need more complex geometrical calculations. These steps will be undertaken in the future.

In the following screenshot you can see the user interface for the tag creation:



Tag Creation

Please fill in the information:

Tag Content (Text):
Road is under construction

Date and time:
07:31:00 AM
Thu, 16 Jun 2005

Tag Validity Period (in Hours): 48

Tag Shape:
● Box

Tag Radius in meters:
○ 10
○ 25
● 50

To create this user interface, we created a class called `TagCreationForm`. It extends the `Form` class and contains the methods, which will return the different parameters used to create a tag. It also reads the user input for the parameters available on the UI. The other parameters are hidden to the user and have default values, except for the location parameter. The location parameter was a problematic issue, which we will explain more in detail in section 5.3. The consequence of this complication was that it was impossible to receive any information about the position from the phone. Hence to make our prototype realistic, we decided to use a number of chosen positions generated randomly.

In order to do that, we first needed to understand how global positioning system works. How is the position measured? Which units are used? A position is made out of two variables: the latitude and the longitude. They give the coordinates on a map. Latitudes and longitudes have the same unit. They are measured with angles from the earth's center to locations on the earth's surface. Latitude measures angles in a north-south direction. Longitude measures angles in the east-west direction. For example, the Hotel des Bergues in Geneva has the following coordinates:

- Latitude: 46° degrees 21' arc minutes 14'' arc seconds – North
- Longitude: 6° degrees 14' arc minutes 71'' arc seconds – East

Degrees, arc minutes and arc seconds are all units to measure latitudes and longitudes. 1 degree is 60 arc minutes. 1 arc minute is 60 arc seconds. 1 degree is thus 3600 arc seconds. As the method provided by Motorola to get a location from the mobile phone uses (arc minutes.10^5) as a unit, we decided to do the same. This will facilitate the transition once the API has been made available.

Having understood how the positioning coordinates function, we started to create a method to generate random positions. To remain realistic, we decided to use positions in the Geneva region. After some searching, we found the position for six well-known Geneva hotels. These positions were of course in the standard format so we had to convert them to the Motorola format. Next, you can see an example of a conversion for the Hotel des Bergues and a table with the hotels and their respective positions:

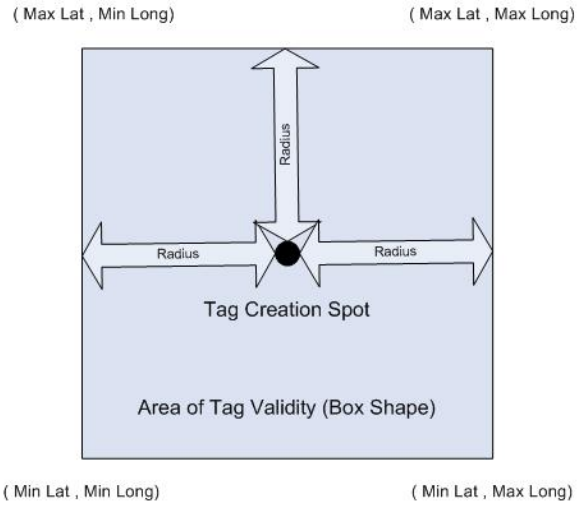
$$\text{Lat. } 46^\circ21'14'' = (46*60\ ?\ 21?\ \frac{14}{60})*100000 = 278068000\ \text{arcmin.}10^5$$

Hotel Name	Position in Degrees	Position in arcmin.10^5
Angleterre	46°20’96” - 6°15’08”	278096000 - 37508000
Bergues	46°20’68” - 6°14’71”	278068000 - 37471000
Edelweiss	46°21’14” - 6°14’98”	278114000 - 37498000
Les Nations	46°21’53” - 6°13’29”	278153000 - 37329000
Mon Repos	46°22’17” - 6’14’95”	278217000 - 37495000
Western Diplomate	46°20’03” - 6’15’98”	278003000 - 37598000

We created two methods[4] called `getLatitude()` and `getLongitude()`. The latitudes and longitudes seen above were stored in an array in their respective method. When the `getLatitude()` method is called, it first generates a random number between 0 and 5. Then the method returns the latitude corresponding to the random number’s position in the array. The `getLongitude()` method returns the longitude corresponding to the same position, as the latitude and longitude we used are pairs.

As tags are not only valid at a single location but over an area, we needed to look at the area of validity of the tag. As we chose to use the box shape, these calculations became easier. Once we have generated the position coordinates, we need to calculate the minimum and maximum latitudes and longitudes. The chart below shows how the area of validity is delimited by those boundary points:

[4] We have used the same method names and declarations as Motorola uses in its location API to limit the number of changes to be made once the location API is available. (see section 5.3)



As latitudes and longitudes are calculated in arc minutes, we needed to find a relation between meters and arc minutes to enable us calculate the coordinates of these determinant points. We decided to neglect the fact that the earth is a sphere and approximate our results:

1km ˜ 30 arcsec

? ?2km ˜ 1 arcmin

? ?2000m?˜ 1 arcmin

? ?1m?˜ 0.0005 arcmin

? ?1m?˜ 50 arcmin.10^5

We now have an approximation of one meter in arcmin.10^5 and can calculate the boundaries of the area of validity. When the user creates his tag, he chooses a radius of validity in meters. The application reads his input. In the `calcLatitude()` and `calcLongitude()` methods, we use the randomly generated location details and that input to calculate the minimum and maximum latitude and longitude of validity:

Minimum Latitude = Latitude – Radius * 50

Maximum Latitude = Latitude + Radius * 50

Minimum Longitude = Longitude – Radius * 50

Maximum Longitude = Longitude + Radius * 50

The two methods associate the minimum and maximum latitude and the minimum and maximum longitude to their respective variables.

On the next page, you can see an overview of the tag format table adapted to our prototype, which shows how the values of the parameters are collected.

Parameter	Value
tagid	Created by the server
imei[5]	Default value: 35491100041572
group	Default value: standard
minlatitude	Calculated according to random latitude and radius chosen by user
maxlatitude	Calculated according to random latitude and radius chosen by user
minlongitude	Calculated according to random longitude and radius chosen by user
maxlongitude	Calculated according to random longitude and radius chosen by user
shape	Default value: Box (Visible to user)
radius	Choice for user on the UI between 10, 25 and 50 meters
activation	Default value: reception
date	Chosen by user on Date Field in the UI
validity	Chosen by user in Numeric Text Field in the UI
priority	Default value: 1
comment	Default value: comment
media	Default value: text
content	Chosen by user in Text Field in the UI

In the main midlet `MobiVTag` class, we have created a `createTag` method, which compiles the different parameters in one `StringBuffer`. The parameters are passed in this format:

(name of the parameter)=(value of the parameter)

The parameters are separated by a carriage return, which helps the server reading the data in a line-by-line stream. `createTag` also uses the networking methods we have created to send the tag. We will look at those methods in the next paragraph.

As there is a lot of information contained in one tag, the parameter method used during our connectivity test earlier on wasn't appropriate. For example, if in the tag content the user entered a space, the server would have problems reading the content. We thus needed to create a

[5] IMEI (International Mobile Equipment Identity) is a unique 15-digit code used to identify an individual GSM mobile station to a GSM network. We use it as a unique identifier.

new method of sending information to the server. In the Tag class, we created the sendHttpPostRequest method connecting with the server during the tag creation procedure. The generic connection framework is still used. However this time we used a POST request to the remote server. We first of all open the connection and send the headers, which tell the server what kind of communication to expect. We open an OutputStream in which we send the tag data contained in the StringBuffer mentioned earlier. If the connection has been successful, the client should receive an HttpConnection.HTTP_OK when calling the getResponseCode method. Next, an InputStream is opened to read the response of the server, which is passed to the calling method.

The sendTag method calls the sendHttpPostRequest by passing the server address and the tag content. It is itself called by the CreateTag method.

To summarize the tag creation procedure:
At first, the tagCreationForm and tag classes are instantiated. The UI is shown on the screen of the mobile device. The user enters the requested parameters and presses the “Send” button. Once the send button is pushed, the information entered on the screen is passed, together with the other parameters to the CreateTag method. This method compiles the parameters into a StringBuffer and calls the sendTag method, which uses the sendHttpPostRequest to send the data. The response of the server is read and passed to a String variable that is used later to show the response of the server to the user.

5.2.3.2 Server Perspecitve

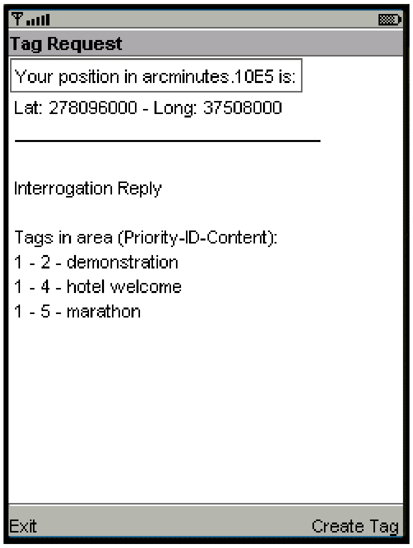
On the server side, we have created a doPost method, which is listening to POST requests from clients. It is responsible for receiving the data from the midlet and storing it. The tag data are stored as Tag objects in an ArrayList. A tag object is defined by its different parameters. When a request is received, we are not using the regular method accessing the parameters using the getParameter method, as the data is sent a block and is read by an input stream. Instead we created an own getParameter method to fulfill our needs. The application reads the BufferedReader line by line and splits the parameter and the value. It then stores the values in a Hashtable corresponding the values and the parameters. The tagid parameter is incremented to give the tag a unique identification. The addTag method is then called to permanently create the tag in the ArrayList with the values for each parameter. Once this has been done, the server sends a reply to confirm the creation of the tag, including a confirmation message, the tagid and the content of the tag.

5.2.4 Third Phase: Tag Interrogation

5.2.4.1 Client Perspective

Apart from creating tags, the mobile device should also be able to interrogate the server concerning available tags at the user's current position. We thus created a standard display, which shows if there are available tags at the current position. In the future, a part of the application should optimize the tag request procedure, depending on the speed of the user, and only interrogate the server when this optimization algorithm deems it necessary. For the moment, we simply use a timer which refreshes the screen every 90 seconds.

First of all, we created a RequestForm class, which handles the user interface of the interrogation phase. It also is the standby view of the application. It shows the currently generated position, as we don't have access to the real location**.** In addition it presents if there are currently available tags. If there are, it shows the priority, the tag-id as well as the content of the tag. On this view, we also receive the confirmation of the tag creation procedure received from the server, if a tag creation request has been made previously. Here follows a screenshot illustrating a regular interrogation view, without the tag creation procedure confirmation:



Your position in arcminutes.10E5 is:

Lat: 278096000 - Long: 37508000

Interrogation Reply

Tags in area (Priority-ID-Content):
1 - 2 - demonstration
1 - 4 - hotel welcome
1 - 5 - marathon

In this class, we have only the information relative to the UI. As we have two versions of the class, we have created two constructors, which have different parameters lists. When we want to post the reply from the server after a tag creation, the response of the server is additionally passed as a parameter and added to the RequestForm. The other parameters, which are present in both cases, are the title of RequestForm, the result of the interrogation procedure and the current details of the position.

In the Request class, we have defined the GET method sendHttpGetRequest, which is used to connect to the server and interrogate it. It's a standard GET request as we used in our first connectivity example. The SendRequest method uses sendHttpGetRequest by handling the URL of the server and the parameters of the GET request, the current latitude and longitude. The latter are generated as in the tag creation procedure. They are at the same time passed to the RequestForm to be shown on the screen when the reply of the server has been received. The GET method returns the reply of the server and is also passed on to the RequestForm through the SendRequest method.

In the MobiVTag class, main class of the program, a timer is used to create a new RequestForm every 90 seconds. When it is created, a new Request object is thus also instantiated and the server is consequently interrogated too. The results are accordingly refreshed on the screen of the mobile device.

5.2.4.2 Server Perspective

On the server side, we created a doGet method, listening for GET requests from clients. When a request is received, the server reads the parameters and parses them into integers. Those integers are passed to the getTags method. In this method, the latitude and longitude will be compared to the minimum and maximum latitudes and longitudes available in the ArrayList. The tags are sorted by priority and by tag id. With the help of an Iterator, the tags are searched one by one. If the latitude is comprised in between the minimum and maximum latitude and the longitude is comprised in between the minimum and maximum latitude of the same tag, the Priority, Tag-ID and content values of this tag are put into a StringBuffer. If more tags are relevant, they are separated by a carriage return. If no tags are valid, a StringBuffer with “No tags available” is created.

This StringBuffer is then passed to the doGet method, which uses it in the reply to the client's request. This reply is sent through an output stream.

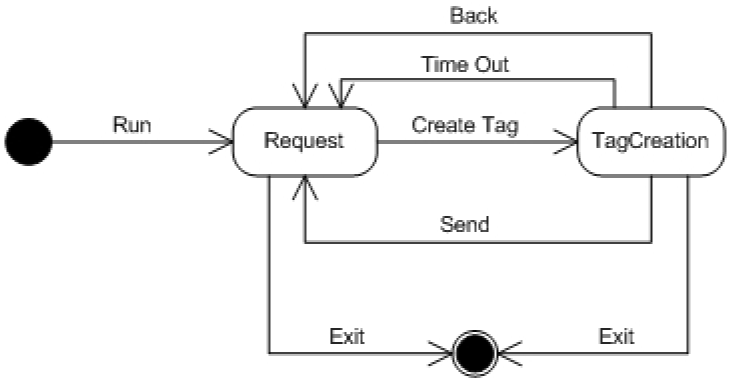
5.2.5 Client Application Overview

We have now seen the different parts of the client application. We still need to get an overview and see how everything interacts.

The main part of the client application is the MobiVTag class. This is the class, which directs the operations. When the application is launched, the startApp method is called. In this method, the Timer of 90 seconds is launched and the TimerTask request is started. This task asks to put the display to the RequestForm. In order to do that, a new Request object is instantiated as well as a new RequestForm. Thus a position is generated, a tag interrogation is sent to the server and the reply is received by the client and shown on the screen.

If the user doesn't use the buttons available, the Request display will simply be renewed every 90 seconds. If the user presses the "Create Tag" button, a new TagCreationForm will be instantiated and the display will be changed to the TagCreationForm. Once the user has filled in the information on the display. He can either press the "Send" button or the "Back button". If he presses the "Back" button, he will cancel his tag creation procedure and simply return to a new RequestForm. If he clicks, the "Send" button, a new Tag object will be instantiated according to his wishes and send to the server. The reply from the server, with the tag creation confirmation message will be received and added to new RequestForm. It will be shown until the Request display is refreshed automatically. The user can at any time press the "Exit" button to quit the application.

The lifecycle of the display object can be seen in this UML state chart, the actions changing the state are either the user pressing the respective button or the tag creation timing out (Time Out):



A UML class diagram of the client application is available in Appendix 5. Appendix 6 and 7 respectively cover the classes used in the tag creation process and in the tag request process of the client application. Appendix 8 shows the classes used by the server application. Appendix 9 shortly presents each class and its purpose. More detailed comments about the programming can also be found in the commented code, in Appendix 10.

5.3 Location API issue

As explained shortly earlier, there were a lot of difficulties getting access to the incorporated GPS in the phone. When we started programming our prototype, we wanted to test the GPS. On the developer guide for the Motorola A1000, provided on Motorola's developer website www.motocoder.com, the chapter 10 explains extensively the use of the Location API. All the methods are clearly stated and their possible utilizations. The guide states how the Location API "consists of the Java package com.motorola.location, which implements location client functionality and of operations for establishing connections to the GPS driver."[6]

So we decided to install the Beta SDK for the Motorola A1000, available on the same website. However, when one tries to use the package, error messages are received. Consequently, we read the User Guide for this SDK. In section 3, Motorola API's, which describes[7], "the various Motorola API's supported in the Motorola handset supported by the SDK", we do not find the Location API. Clearly this SDK doesn't support the use of the Location API.

When searching the web for answers, we came across persons on forums, with the same problems. Here follows an example of these testimonials: "Anyone knows where to find the fully licensed java SDK for Motorola phones? The one that is available for download from motocoder.com doesn't contain the location classes so it's impossible to acquire information from the agps-receiver in the phone."[8]

We decide to further investigate the question with Motorola. On their motocoder website, Motorola offer a 24/7 tech support. We state a clear question, explaining our project and receive the following answer:

[6] Page 32, Developer Guide Motorola A1000

[7] Page 7, Motorola SDK A1000

[8] Svante Johnson, 14.02.2005, Online resource available at: http://www.3g.co.uk/3GForum/archive/index.php/t-15809.html

"Only developers with a business relationship with Motorola can access closed (or licensee only) APIs. To be evaluated for a business relationship with Motorola please refer to the below." Shortly, Motorola ask to send a description of the project, what kind of company we are and similar company related questions. This request should be made to innovate@motorola.com, allowing 4 to 6 weeks for evaluation. At this day, we haven't received any reply after two requests in 10 weeks.

As we didn't receive any reply, we tried a second time through the tech support site of Motorola[9]. We state the same questions as previously, but receive a totally different answer:

"If you are requesting the JAVA API for the Location API please follow the proceeding procedure:

Should you have access to an A1000 from the Hutchison 3G network and would like access to the A-GPS Location API from within J2ME, then it is recommended that you seek guidance from the Operator "Hutchison 3G". The reason for this is that the API is protected under a security signing framework and requires H3G authority to access this feature."

As we have bought our mobile through 3, which is the same as Hutchinson 3G network, we kindly ask Motorola to give us a precise contact at 3, to help us receive a quick answer. Unfortunately, Motorola reply negatively and pretend not to have any contact details of 3.

Having had previous contact with 3, before buying the phone, we sent immediately an email to their customer service. However they have deactivated this e-mail address and all requests have to be made through their website interface. A request has been sent through their website. Until today, this request has not been answered.

Searching for alternative solutions, we found a navigation program using the GPS on the Motorola A1000. When testing the trial version, the GPS is indeed activated. We tried to contact the producers of this application to seek for guidance, but they were enabling to help us. According to them, the SDK for the A920, a precursor to the A1000, had the location API included. Another website[10] offers a free GPS solution for the A920 and A925, with open source code. We tested it on the A1000 but it was unfortunately incompatible.

Apparently, Motorola, in accordance with 3, have decided to change their policy and closed down development opportunities. Without the help of 3 is thus impossible to use the location API and access the working

[9] Available online at https://motocoder.custhelp.com

[10] Available online at http://per.nitro.dk/

GPS on the Motorola A1000. This caused our decision to use randomly generated provisionally enabling us to create a partly functioning prototype. However, we used the same method names and declarations as used by Motorola in the A1000 Developer guide to limit the number of changes to be made once the location API is available.

5.4 Future Developments

The main task for the future is to get access to the location API from Motorola by acquiring the licensed SDK, which includes the different packages. This is essential for the continuation of the project. The system designed is a location-based service. If it is impossible to get the location details, it doesn't fulfill the purpose of its existence. Additionally, if the program can be created using the Motorola SDK, the getMachine method to get the IMEI of the phone will be able to be fully implemented.

A key evolution will be to change the interrogation request procedure from a timer, as it is now, to an intelligent agent. It will have to calculate when it is appropriate to send a new tag interrogation, according to various parameters, as for example the speed of the user and relative change of position.

The tag validity feature will have to be worked on. It should become possible to give an unlimited time of validity. However if a precise date and time is given the tag validity needs to be translated from an amount of hours in a date and time item according to the time of creation of the tag and the validity period as chosen by the user. The server will need to deactivate the tag once this date and time item has been reached, without deleting the tag.

The application also needs to be able to receive other information than text as content. It should be able to read different MIME data types. How to implement this and how to create other types of data are challenges that required to be met.

The vibration feature of the phone is important, as we are confronted with visually impaired. A definition of its use needs to be made.

On the server side, the use of a database will have to be implemented, connecting the server to the database using JDBC. The data in the database must be checked for consistency. Furthermore, as the system is to be used by more than one client, the server has to be optimized. It needs to be powerful and available at all times. The scalability has thus to be improved. Security and privacy issues have to be studied thoroughly.

The tag archive will also become a very useful resource in the future. The analysis of the tag history could be used in predictability researches and also for other purposes.

Software wise, it would also be good to have the midlet verified in order to receive the Java Verified label, ensuring it is a trusted midlet. This will take away the "untrusted midlet" warnings during installation and when the midlet makes its first network request. The midlet would then also need to be able to be installed over-the-air.

We have seen a variety of propositions for future improvements of the MobiVTag project. However there are many more. We need to keep in mind that the software's end-user will be visually impaired persons. Thus, as targeted users, the particular requirements of visually impaired necessitate to be studied closely in the future. They know theirs needs better than anyone. Working hand in hand with them will ensure a further success of the MobiVTag project.

Chapter 6 – Conclusion

The MobiVTag project started with a project proposal. It has since then evolved significantly. Scenarios of use have been prepared. The architecture of the system has been modelised. A long period of time has been put in choosing the appropriate hardware and a first prototype application has been programmed, opening opportunities to continue with the future steps of the project.

However, this mémoire has helped us discover a more important issue. The problems encountered during the buy of hardware and especially the use of the mobile phone once it had been acquired raise deep concerns.

The disparity of technology was our first observation. How can Motorola motivate that their newest UMTS phones are not available in Switzerland, although the UMTS network is available and functioning? Motorola was unable to provide or sell to us directly one of their existing mobile phones for a purely non-commercial research project, conducted by a renown European university.

When starting the programming, it soon became clear the key programming features of the phone were unavailable and Motorola were unable to give us access to these API's: Either not responding to our email requests or blaming a security signing framework, requiring the 3 network authority to access the requested feature. The 3 network were not of much more help as our emails remained unanswered. Clearly, innovation in the telecommunication field is blocked by collaboration between network operators and mobile phone producers. By requiring special license agreements for certain API's, the telecom industry control development and refrain the possibilities of pursuing innovative ideas from outside parties. Having paid enormous amounts of money in a period of wild bidding to acquire UMTS licenses in European countries, mobile operators want to guarantee their income by controlling as much as possible. Obviously, location based services have been identified as resources to control.

The consumers have simply been once again taken hostage by the telecom industry, as it is the case with the prices of overall phone communications. In a broader perspective, innovation can be seen as an essential element for the growth of our economies and for the evolution of our society. However these effects are slowed down when major industrial players refrain the right to innovation. If the benefits to the society in a wide sense should prevail over the benefits of only shareholders, something has to be done to safeguard the right to innovation.

Bibliography

Dimitri Konstantas, *Mobile Virtual Tags for the Visually Impaired* (2004).

Julio Sanchez, Maria P. Canton, *Java 2* (2000), IDG Books, 1st Edition.

Walter Savitch, *Java, an Introduction to Computer Science and Programming* (2001), Prentice Hall, 2nd Edition.

Steven Holzner, *Java Server Pages* (2002), SAMS, 1st Edition.

Martin de Jode, *Programming Java 2 Micro Edition on Symbian* (2004), Symbian, 1st Edition.

Motorola, *Motorola A1000 SDK Users Guide* (2004), Motorola, 1st Edition.

Motorola, *Motorola A1000 J2ME Developer Guide* (2004), Motorola, 1st Edition.

Chantal Morley, Jean Hugues, Bernard Leblanc, *UML pour l'analyse d'un système d'information* (2002), Dunod, 2nd Edition.

Pascal Roques, Franck Vallée, *UML en action,* (2003), Eyrolles, 2nd Edition.

Online: http://doc.trolltech.com/4.0/model-view-programming.html, Trolltech, 2005.