

9<sup>th</sup> semester project – 2008/2009

Specialization in Vision, Graphics and Interactive Systems

Department of Electronic Systems – Aalborg Universitet



VGIS  
GROUP 920

Bus Router

A route planner for the bus user in the city of Aalborg

Henri LEGENTIL;Thomas LUEL;Florent MAZZONE;Sergio PEDRO;Thomas ZUBELI

**DEPARTMENT OF ELECTRONIC SYSTEMS****INTELLIGENT MULTIMEDIA 9TH SEMESTER****TITLE:**

BusRouter

**THEME:**

Intelligent Multimedia

**PROJECT PERIOD:**September 5<sup>th</sup> 2008 to  
January 5<sup>th</sup> 2009**PROJECT GROUP:**

08grp920

**GROUP MEMBERS:**

Henri     Legentil

Thomas   Luel

Florent   Mazzone

Sérgio     Pedro

Thomas   Zubeli

**SUPERVISOR:**

Lars Bo Larsen

**NUMBER OF COPIES:**

3

**REPORT PAGES:**

138

**APPENDIX PAGES:**

4

**TOTAL PAGES:**

152

**SYNOPSIS:**

This report documents the development of a mobile navigation aid for Bus Passengers. The system developed helps a bus user to get and follow the fastest path to its destination using walk and bus with his mobile phone. The system was developed using a client/server architecture:

- A server used to retrieve the bus schedule and calculate the shortest path between the departure point and the arrival point.
- A mobile phone application used by the user to enter his destination and arrival addresses, to visualize the path to follow on a map and to listen to the audio guidance information.

The real time bus schedules are retrieved from the NT Live Bus server, the map from OpenStreetMap server.

The whole project has been developed in Java programming language. The mobile phone application has been developed using the Nokia S60 emulator and tested on a N95 device.

This report presents the systems development life cycle from the analysis to the tests.

# Preface

This report and all of its contents are part of the 9<sup>th</sup> semester in Vision, Graphics and Interactive Systems at Aalborg Universitet. It has been written by the project group VGIS 920 during the fall semester of 2008.

The semester theme is: Analysis and Synthesis of Images. Its purpose is to introduce the students to advanced theories and techniques within the core disciplines of the specialization. The semester is divided into some courses, from which we took some theories and techniques to apply to our project. The theme of our project is: Mobile Navigation aid for Bus Passengers.

Each chapter and the corresponded sub chapters are marked by one or more consecutives numbers. As more consecutives numbers has the index, more deeply is the information present in there. The same works for the images.

## Report outline

---

The structure of our report is divided into 7 parts:

1. Analyses: here we analyze all the different technologies and techniques that we thought useful for our project. For instance, we analyzed localization systems, operating systems, map providers, algorithms and mobile communication.
2. Design: here we explain how we designed all of the components of our projects, either in a global view or in a more specific view.
3. Implementation: here we explain which problems we faced during the code's phase. We also explain the choices that we made to avoid them and which problems remained without a possible solution.
4. Testing: here we show the final tests that we made to our final program and which results we obtained.
5. Conclusion: we give in this part a brief conclusion of all of the project, comparing the early ideas of the project and the final solution
6. Bibliography: we give a list of references that we used to develop our project and to write this report.
7. Appendixes: in this section we included some tables, graphics and example files that are not so relevant to include in the previous parts of the report.

## Authors

---

The VGIS 920 team, composed by Henri Legentil, Thomas Luel, Florent Mazzone, Sérgio Pedro and Thomas Zubeli.

# Acknowledgments

We would like to thank all the people who helped us during the accomplishment of our project, and particularly:

- Our supervisor Lars Bo Larsen, semester coordinator and project supervisor for his help, management and advice.
- All the course lecturers, for their advice and the courses that helped us to develop our project.
- Ms Charlotte Skindbjerg Pedersen, semester secretary, for her support regarding the administrative issues.
- M. Mejdal Rasmussen, from the IT workshop, for his help and support during the installation of our project.

# Table of contents

List of figures .....	1
Introduction.....	4
Analysis .....	7
1) Mobile phones .....	8
1.1) Application running ability.....	8
1.2) Web connectivity .....	9
1.3) GPS Access .....	10
2) Localization system .....	11
2.1) Mobile Phone Networks Localization.....	11
2.2) Network assisted solution .....	16
3) Map .....	16
3.1) Nokia Map.....	16
3.2) Navteq .....	17
3.3) OpenStreetMap .....	18
3.4) Google Map .....	18
4) Shortest Path Algorithms .....	20
4.1) Dijkstra's shortest path algorithm.....	20
4.2) Restricted Search Algorithm .....	20
4.3) A* search .....	23
5) Application Server.....	24
5.1) Unix-core based operating system.....	24
5.2) Windows-core based operating system .....	24
5.3) Conclusion .....	25

6) Bus schedule .....	25
6.1) Fixed schedules .....	25
6.2) Live schedules .....	26
Design.....	29
1) Structure.....	29
1.1) Global Diagram .....	29
1.2) The GPS interactions .....	31
1.3) The Mobile Phone interactions.....	31
1.4) The Application Server Interaction .....	32
1.5) The NT Server.....	33
2) Mobile phone application .....	34
2.1) Destination Mode .....	34
2.2) Map mode.....	46
2.3) Favorites and Settings .....	46
2.4) User interface .....	47
2.5) Programming language choice.....	85
2.6) Code structure .....	86
3) Server .....	91
3.1) Map .....	91
3.2) Geocoding.....	101
3.3) Bus schedule retriever.....	107
3.4) Shortest path calculation.....	112
4) Interaction Server / Mobile phone .....	120
4.1) Exchange protocol.....	120
4.2) XML file.....	121
Implementation.....	125

1) Mobile phone .....	125
1.1) Issues met during the development.....	125
2) Application Server.....	128
2.1) Schedule Retriever .....	128
Testing.....	135
1) List of tests.....	135
2) Results of the tests.....	135
Conclusion .....	139
1) Project Outcome .....	139
2) Personal Achievement .....	140
3) Perspectives.....	140
References.....	141
Appendixes.....	143
4) Class diagram of the mobile phone and server application .....	147





# List of figures

Figure 1 : Simplified diagram of the system.....	7
Figure 2 - Operating systems market shares .....	8
Figure 3 - Java program example .....	9
Figure 4 - Windows mobile phone .....	9
Figure 5 - iPhone 3G .....	10
Figure 6 - External GPS receiver .....	11
Figure 7 - Localization using triangulation .....	13
Figure 8 - Trilateration using 4 GPS satellites .....	15
Figure 9 - Nokia map GUI example 1 .....	17
Figure 10 - Nokia map GUI example 2 .....	17
Figure 11 - Search Area of Dijkstra and Restricted Search Algorithm .....	21
Figure 12 - Check point in polygon .....	22
Figure 13 - Bus schedule example .....	26
Figure 14 - Live schedule example .....	27
Figure 15: Global Diagram (elements in purple are the ones we will develop ourselves, elements in green are optional) .....	30
Figure 16 - user to point distance.....	35
Figure 17 - First problem faced .....	36
Figure 18 – Modify from intmath.com .....	39
Figure 19 - Second problem faced.....	40
Figure 20 - Checkpoints .....	42
Figure 21 - distance to the end of the segment.....	43
Figure 22 - Answer to the question: "Application is easy to use" .....	57

Figure 23 - Answer to the question: "Design user-friendly.....	58
Figure 24 - Answer to the question: "All features are pertinent and useful" .....	58
Figure 25 - Answer to the question: "Interfaces logically organized" .....	59
Figure 26 - Answer to the question: "Application is easy to use" .....	70
Figure 27 - Answer to the question: "Design user-friendly.....	71
Figure 28 - Answer to the question: "All features are pertinent and useful" .....	71
Figure 29 - Answer to the question: "Interfaces logically organized" .....	72
Figure 30 : Tree of the mobile phone application screens and their relations to each other. .	74
Figure 31 : Main menu of the mobile phone application .....	75
Figure 32 : Main menu after rotation to the right. The user can press OK to view the map. ..	75
Figure 33 : The settings menu.....	76
Figure 34 : The user is editing the parameter "sound". .....	77
Figure 35 : There are no favorites created yet.....	78
Figure 36 : Creation of a new favorite. ....	79
Figure 37 : Favorites List and the available options. ....	80
Figure 38 : Edit the favorite // Pop-up the user gets just after selecting the "delete" option.	80
Figure 39 : Map of the surroundings, with the position of the user marked on it.....	81
Figure 40 : Start from current position or Start from an address. ....	82
Figure 41 : Destination Path screen.....	83
Figure 42 : Text Mode.....	84
Figure 43 - Bus Line Creation .....	93
Figure 44 Map class diagram.....	95
Figure 45 Information processing .....	95
Figure 46 Way to study .....	99
Figure 47 : Calculation of the distance between U and A, B, C.....	103

Figure 48 : The ways W and X will be selected. ....	104
Figure 49 : The point C is the nearest, but the way W is nearer. ....	105
Figure 50 : the distance to the segment is the distance to the point P. ....	106
Figure 51 : The Segment [AB] is selected. The nearest point will be the nearest between A and B .....	106
Figure 52 Map analyzed by the algorithm .....	114
Figure 53 Shortest Path on the map. ....	119
Figure 54 - BusChecker's flow .....	129
Figure 55 - Retrieving the information from different sources. ....	130
Figure 56 - The problem of the offsets .....	131
Figure 57 Error introduced by the absence of the bus stop arrival time at a bus stop .....	131
Figure 58 - Final structure of the bus schedule retriever .....	132

# Introduction

## Aim of the project

---

Nowadays, the navigation systems are more oriented at in-vehicle navigation. In fact, except for hikers, the navigation systems for pedestrians are not very well developed. One of the main reasons for this is the accuracy and the signal strength needed for these navigation systems. For instance, a pedestrian user can enter on ways between building, inside parks, inside buildings, etc.

Another important thing, and which is one of our project's goals, it's about the public transport. With the great development of the vehicle industry, happened also a great development of the transports (bus, train, subway and tram) for city's users. Thus, many times, the user wants to go to some place and doesn't know which transport to take him there or to which station he has to go to take his transport. This can be a very important market for navigation systems in the future.

In Aalborg, the most used kind of transport, besides the car and the bike, is the NT bus. NT is a bus company. They run all the buses in Aalborg. Since the NT bus network is quite complex, it can be a very good proof concept to develop a navigation system for bus users. Thus, we are getting to the point to explain why this project seems important for us. So, this project should combine two kinds of navigation:

- Pedestrian navigation
  - o From the start point given by the user to the bus stop that he has to go to take the bus.
  - o From the final bus stop to his destination.
- On-bus navigation
  - o Giving the bus stop that the user has to get off to change to another bus.
  - o Giving the bus stop that the user has to get off to do the final pedestrian way to his destination.

Since these objectives can only be achieved on mobile devices, we needed to develop a mobile phone application to be installed on the user's mobile. This mobile phone application has to frequently retrieve the position of the user, using localization systems, such as GPS. Besides that, the mobile phone application needs to implement a Graphical User Interface (GUI) for an easy use of the application. Another thing needed is the map provider for the mobile application (e.g. Google Maps). Finally, we have to analyze which devices we are going to develop our mobile application for, since there are many of them in the market with different operating systems.

As you probably know, a mobile device is a computer by itself, but not as powerful as a home computer. For this reason, we can't expect all the calculations to be made by the mobile. We need another application running on a server machine, that we'll call application server. This application will be responsible to communicate with the user's mobile device, retrieve the user's coordinates or an address, do all the necessary calculation and in the end return it to the mobile application. For this, we'll need to analyze also some algorithms, to handle a good shortest path calculation, and also

how to handle with the coordinate systems, that means, how to connect streets, roads, bus routes, bus stops, etc.

Concluding this short introduction of our project, here are the main objectives that we propose to do achieve in the early beginning of this semester:

- Research into mobile navigation and mobile phone programming technologies.
- Development of client software in a suitable language such as Java.
- Development and evaluation of user interfaces.
- Use with or without GPS device (the user can provide his/her start address manually).
- Best route calculation for walking and through the bus network.
- Can calculate the users expected departure time, given the desired arrival time, place and departure location.
- Gives directions when walking to bus stop and warns if user takes a wrong turn.
- It's possible to save some addresses.
- The user interface is in 4 different languages.
- Warns the user when it is time to leave the bus.
- Implements a simple, easy to learn user interface.

Since now we presented our project, we'll give an example a possible scenario of a normal bus user in Aalborg using the *BusRouter*.

## Scenario

---

For an example user, let's call the user John. John is an Erasmus' student that arrived Aalborg just one day ago. John lives on Danmarksgade and will start the lectures tomorrow on Aalborg University. Before coming to Aalborg, John installed on his mobile device the application *BusRouter*. He heard that gives help for the bus users in Aalborg, and since he doesn't have car or bike, he will use the Aalborg city buses during the next months.

- John wants to go to Aalborg University. It is the first day of lectures for him in Aalborg University. He is getting ready to get off from his flat, when he realizes that he doesn't any idea which bus to take to the University. Instead of asking to someone in the street and losing some time searching for the bus stop, he just starts his mobile application *BusRouter*. In the first menu, there's an option called *Destination*. John decides to press **ok** and goes to second menu where are all the text fields to be filled to get his route. Fortunately, he has the address of his department that is Niels Jerns Vej 12. He puts all the address in the text fields and presses **go**. After this step, John has finally his route on the mobile screen.
- John checks that he has to go to Politigarden to take the bus number 2. Checking the route, John follows what he thinks that is the correct street. But he goes in a wrong street, so the *BusRouter* recalculate a new path for John and advices him that is going in a wrong direction. Finally, John goes to the right street and in the end he arrives to the bus stop Politigarden.

- After 20 minutes inside the bus, John reads on the screen that the next bus stop is the one that he has to get off the bus. John presses the red button of the bus and finally gets off the bus. In the end, John follows the rest correctly and arrives to his department, where asks where is it the lecture.
- In the next day, John wants to go again to the University. He remembers most of the route to there, but since he went yesterday in the wrong street, he wants to know again the correct way until the bus stop. So he starts the *BusRouter* application, and once he is looking on the first menu. He thinks “Why each time I have to put all the address in the text fields?”. He thinks that there is a better way, like saving this address in some favourites menu, like the web browsers, where is possible to save an URL, by giving it a name that we are familiar. So, he press one of arrow keys a couple of times and appears written on the screen *Favourites*. John presses **ok** and goes to a new menu where it is an empty list of favourites. He presses **options** and after **new**. John fills all the text fields and in the end he saves with the name **University**. After this, he enters again in the *Favourites* menu, and over the label **University** he presses **go to**. After this, he has again the route displayed on the mobile phone’s screen. This time, John follows the correct path and arrives after half an hour to the University.
- In the end of the day, after arriving home, John wants to navigate a little bit on the map of Aalborg. Since he doesn’t have any application to do that, he is wondering if the *BusRouter* has such a support just for map view. He starts once more the *BusRouter*, and after rotates the main menu with the arrow keys, on the mobile phone’s screen appears **view map**. He presses **ok** and he has the map of Aalborg displayed on the screen. He navigates a little bit through the map, discovering a little bit of the geography of Aalborg, that it can be useful for him in the future.

This was a possible scenario of use of the *BusRouter*. In the next chapter we’ll describe in detail all the features that we analyzed to do both applications (Mobile application and Application server).

# 1 Analysis

In order to fully understand our project, we thought it was necessary to have knowledge about some technologies. That is why we are going to present each technology, domain, or field of knowledge that we had to study before we could start to design the system.

First, we have to show very briefly why we chose to explain those elements. The figure below is a very simple diagram showing the different part of the project. The mobile phone is connected to a server, which retrieves information on NT server and calculates the shortest path. We can also note that the user interacts with the mobile phone. The mobile phone may contain a GPS, use one, or rely only on the mobile network to localize the user. The mobile phone we worked on was a Nokia N95, and the project was designed for the series S60.

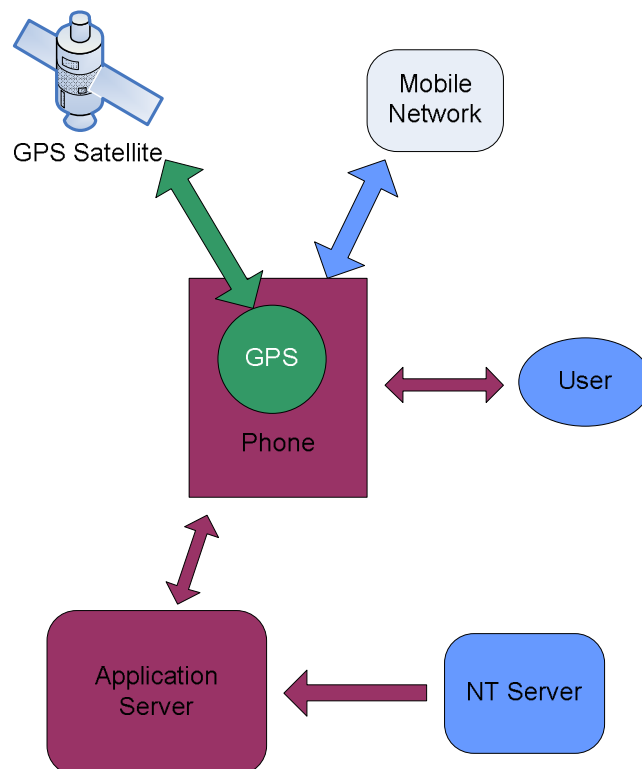


Figure 1 : Simplified diagram of the system.

We will now present our researches on each element separately.

## 1) Mobile phones

First of all, we are developing an application for mobile phones. So the main pre-requisite is that the mobile must support at least one platform which can execute applications. It might also be able to connect on the web. Indeed, the mobile phone has to connect to our server. And finally, to use the navigation feature, the system should retrieve, by one way or another, its geographical coordinates. This last point is not mandatory, but that is the reason why we are going to have a look at the GPS availability on mobile phones.

### 1.1) Application running ability

Nowadays, most of the mobile phones are able to run small programs. As the program we were aiming to develop had to be embedded, it has to respect the constraints of these mobile phones and stay within their limitations. We are going to talk later about the choices we made about the programming language, but at the very beginning, we studied all the different possibilities.

The most commonly used is Java. The games downloaded on mobile phones represent the biggest percentage of the applications, and are almost every time coded in Java. When developing an application in Java for a mobile device, it's the J2ME (Java Mobile Edition) that it is used. It represents a real potential solution to our problem area. More and more, new devices released are Java compatible.

Let's now have a look to the operating systems:

<b>Worldwide smart phone market Market shares Q3 2008, Q3 2007</b>					
<b>OS vendor</b>	<b>Q3 2008 shipments</b>	<b>% share</b>	<b>Q3 2007 shipments</b>	<b>% share</b>	<b>Growth Q3'08/Q3'07</b>
<b>Total</b>	<b>39,850,100</b>	<b>100.0%</b>	<b>31,156,240</b>	<b>100.0%</b>	<b>27.9%</b>
<b>Symbian</b>	18,583,060	46.6%	21,219,390	68.1%	-12.4%
<b>Apple</b>	6,899,010	17.3%	1,107,460	3.6%	523.0%
<b>RIM</b>	6,051,730	15.2%	3,298,090	10.6%	83.5%
<b>Microsoft</b>	5,425,470	13.6%	3,797,360	12.2%	42.9%
<b>Linux</b>	2,028,490	5.1%	1,361,810	4.4%	49.0%
<b>Others</b>	862,340	2.2%	372,130	1.2%	131.7%
<b>Source: Canalys estimates, © canalys.com ltd. 2008</b>					

Figure 2 - Operating systems market shares

On one hand, a certain number of mobile phones, or better, personal assistants, run Microsoft Windows Mobile. As it is more complete and more business oriented, the users are mainly the businessmen, through their professional use. More than only the devices, Windows can provide a full enterprise management solution, including both client and server side. This makes companies be



more likely to choose this structure. [i - ZDNet, 2008] On such devices, you can still use Java, but the main purpose is to offer the access to the Microsoft .NET Environment. You will then be able to run programs coded in (C#, VB, etc.).



Figure 3 - Java program example



Figure 4 - Windows mobile phone

On the other hand, we have Symbian OS. This is without any doubt the most widespread operating system on the market: in terms of devices, it represents about 70% of the shares. [ii - PDAsNews, 2007] This is, for a part, thanks to the Nokia share on the market and its unique policy of Symbian-only range of products.

The difference between these two leaders in the smart phones systems is narrowing more and more. To make a short comparison, in respect to the relevant criteria for our project [iii - LetsGoMobile, 2008]:

- Stability: Symbian is more stable than Windows Mobile, even if the last one has clearly improved in this domain.
- Energy: once again, Symbian has the lead. By energy, we mean the consumption of energy, when using demanding features, such as Bluetooth, WiFi and GPS

These are the two systems, able to execute applications, which are the most widespread among the mobile phones on the market. Besides that, you will find other platform generally less known such as or Palm OS, Blackberry's OS Linux, or iPhone's OS.

## 1.2) Web connectivity

In November 2007, a survey has been made in 29 countries, interviewing 16.000 customers equipped with mobiles phones. It aimed at showing the motivations and restraints with regard to mobile use. [iv - TNS-Sofres, 2008]

According to this survey, 41% of the mobile phones, all over the world, are equipped with a web mobile connectivity technology. However, only 20% of the owners of these devices use this feature. In Europe, it's more widespread. Indeed, more than two thirds of the devices are compatible, but only 18% of the users use it.

According to the managers of main companies, this is not a matter of equipment, but more likely due to the lack of offers from the telecom companies. Indeed, the communications through the web are not often enough included in mobile plans, and it is quickly really expensive to use it. But, hopefully, the market experts expect the mobile network providers to align their offers with the land Internet providers' unlimited offers.

Other reasons to this slow adoption of the mobile web seem to be, still according to the survey, the size of the screen and the keyboard ergonomics while browsing web pages. These problems seem not as relevant as the rest for our project. It uses the connectivity to the network, but the only moment when the user would have to interact directly with it would be to enter the address, unlike when surfing on web pages.

### 1.3) GPS Access

The survey we talked about earlier gave us one more important number: one third of the users of mobile phones would be interested in the geo-localization technology, which is a lot.

This means that our product, once finalized, will answer to a real need, that users express.

But at the moment, only high range devices are fitted with an internal GPS module. The new iPhone 3G has been a new step in the democratization of GPS equipped mobile phones. It's not really recent, but it is still has been one of the last improvements brought to smart phones.



Figure 5 - iPhone 3G

An alternative solution to this situation has been the use of external (mainly Bluetooth) devices. Indeed, the Bluetooth has been implemented earlier, by the time when the use of the mobile phone as a GPS was not well-established.



Figure 6 - External GPS receiver

## 2) Localization system

---

As the mobile phone has to locate the user prior to any route calculation, the localization is a main issue. It has to be:

- accurate enough (street width)
- fast to synchronize
- reliable
- compatible with the project language (java/C++ application programming interface needed)

Usually, mobile phone manufacturers implement a *GSM Localization* interface to help the mobile network operator to locate their users. Mobile Phone Triangulation and Cell identification are two GSM localization options. Five years ago, a new localization technology already used in the transportation sector, the *Global Positioning System* (GPS), started to be implemented in several mobile phones.

What are the advantages/drawbacks of these different solutions?

### 2.1) Mobile Phone Networks Localization

To improve the audio quality, telecom providers need to use the GSM base station which is closest to the users. Knowing the distance between the user and the antenna implies that the network operator is able to locate the user.

In order to calculate the position of a mobile phone, there are several different methods using the service provider infrastructure and/or the user's device. We can differentiate three solution families:

### ***2.1.1) Network based***

The network is able to process some basic information by sending and receiving data to a mobile phone. [v - Eric Vialle, 2008]

#### **2.1.1.1) Cell identification**

Network assigns a cell id to each antenna. When the user turns on his mobile phone, it detects the nearest antenna. The network localization system uses this id to find the localization of the antenna and is able to give an approximate position of the mobile phone. (In a circle around the antenna)

**Advantages:** fast (2-3s), secure, no software installed on the handset

**Drawbacks:** poor accuracy (depends on the cell radius  $\pm 0,250\text{km}$  city < ... <  $\pm 10\text{km}$  rural area)

#### **2.1.1.2) Triangulation**

When the user turns on his mobile phone, it detects the nearest base station (at least 3). The network localization system uses these antennas to send some service messages to the mobile phone which answers to each antenna. The answers reception times are then processed by the network system in order to calculate the distance between each antenna and the mobile phone and estimate the mobile phone localization.

**Advantages:** fast (5-8s), secure, no software installed on the handset, more accurate than cells identification.

**Drawbacks:** medium accuracy ( $\pm 0,125\text{km}$  city < ... <  $\pm 4\text{km}$  rural area)

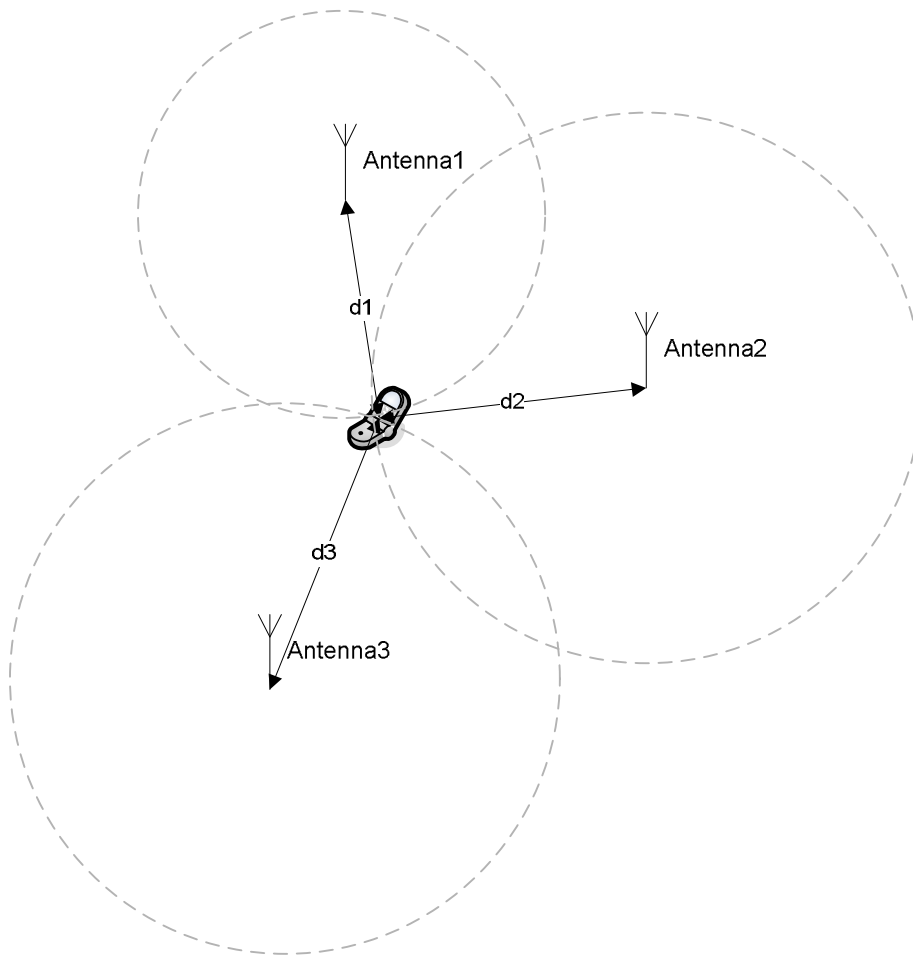


Figure 7 - Localization using triangulation

### 2.1.2) Handset based

As the network based methods calculate a rough estimate of the device location, phone manufacturers nowadays implement directly in the mobile phone an independent and precise localization system. In order to check its location, the mobile phone can use the network operator installations such as antennas (cell identification, wimax, triangulation...) or any other means such as satellites (Global Positioning System).

#### 2.1.2.1) Global Positioning System (GPS)

In the section, we are using several reference, which are: [vi - Kowoma, 2005], [vii - Deblauwe, 2008]

#### What is the GPS?

The NAVSTAR GPS is a **Global Navigation Satellite System** (GNSS) project created by the United States army in the sixties. The first developmental satellite was launched in 1978. Even if numerous similar projects were launched since its creation (Russia → GLONASS, China → Beidou Navigation system, Europe → Galileo), the GPS becomes in 1995 the only fully functional GNSS in the world.

The system is based on a constellation of earth orbit satellites (24 to 32) used to emit radio signals. These signals processed by a GPS receiver on the earth enable it to check its distance with each satellite and then check its location, altitude, speed and the Coordinated Universal Time.

### GPS Signal

GPS satellites emit signals on numerous frequencies depending on the users. Civilians can decode the 1575 Mhz signal that includes a navigation message and coarse/acquisition code:

- Navigation message (50 b/s): Each satellite sends periodically a frame containing the time of weeks, the GPS week number, the satellites status, an ephemeris (satellite orbit) and an almanac (orbit and status information of the satellites, data used to correct approximation errors).
- Coarse/acquisition code ( $\sim 1$  Mb/s): Each satellite sends a specific frame of 1023 bits with a period of 1 ms enabling the receiver to identify the satellite.

### Calculation of the distance between the satellite and the receiver

Using the navigation message and its own time, the receiver is then able to calculate the signal of each satellite and to compare the emission time with the reception time. The difference is called the propagation time ( $\Delta t$ ). As the speed of the signal is approximately equal to the speed of light ( $c = 299\,792\,458\text{ m.s}^{-1}$ ), we can deduce that the distance to the satellite is equal to:

$$d = \frac{\Delta t}{c} \text{ in meters}$$

### Calculation of the position

To determine a precise localization, a receiver must track at least four satellites in order to synchronize its clock and acquire the satellites distances. Using a set of equations, the receiver is then able to determine its position by measuring the intersection point of three spheres (radius = the distance of each satellite).

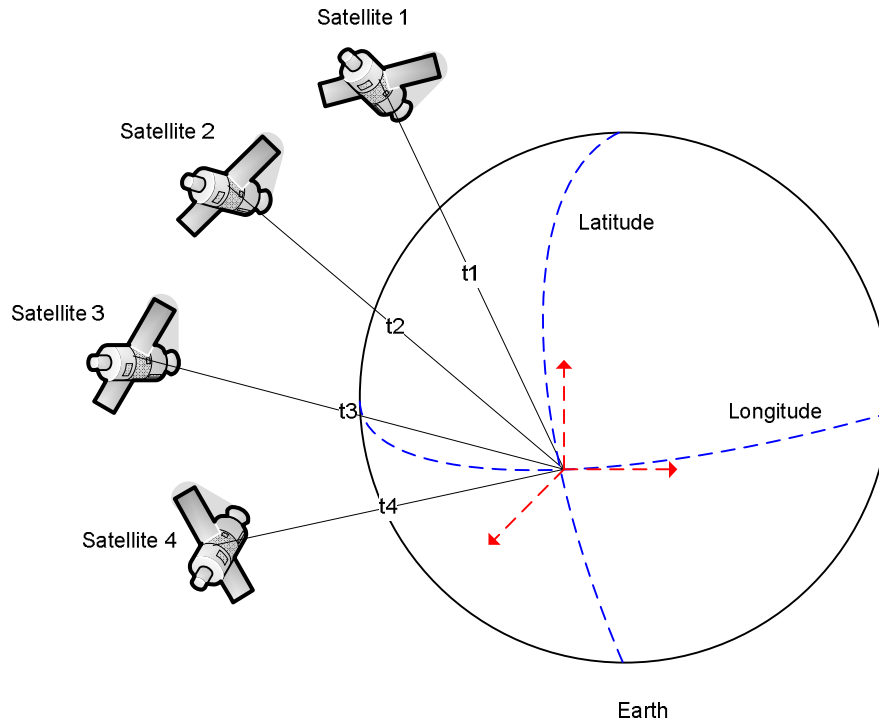


Figure 8 - Trilateration using 4 GPS satellites

### GPS data format

Nowadays, most of the GPS data receivers use the NMEA protocol to communicate data with others devices. The NMEA 0183 standard use a ASCII, serial communication protocol that give the principal information about the GPS signal such as the localization, altitude, time or quality of the signal.

### An example of NMEA sentence sent by a GPS receiver and its signification

*\$GPGGA,171614.000,5702.2685,N,00956.2351,E,1,04,2.1,111.8,M,42.5,M,,0000\*53*

<i>\$GPGGA</i>	: Global Positioning Fix Data
<i>171614.000</i>	: Time (UTC)
<i>5702.2685</i>	: Latitude
<i>N</i>	: North or South
<i>00956.2351</i>	: Longitude
<i>E</i>	: East or West
<i>1</i>	: GPS quality indicator (0 fix not available, 1 GPS fix, 2 Differential GPS fix)
<i>04</i>	: Number of satellites in view
<i>2.1</i>	: Horizontal Dilution of precision
<i>111.8</i>	: Antenna altitude
<i>M</i>	: Units of antenna altitude (meters)
<i>0000*53</i>	: CheckSum

**Advantages:** accuracy ( $\pm 10\text{m}$ ), cheap, very good accuracy in rural zones

**Drawbacks:** nowadays few mobile phones have a GPS, skyscrapers can blur the GPS signal, synchronization time (often more than 1 minute), unavailable in buildings, power consuming.

## 2.2) Network assisted solution

To explain what follows, we relied on theses references: [viii - Scott Pace, 1994], [ix - Openwave, 2002], [x - Shu Wang, 2008]

We learnt in the previous parts that network based localization methods are fast, run almost everywhere and offer a poor accuracy whereas the GPS (handset based) is precise but slow to synchronize. In order to use the advantages of these methods, mobile phone manufacturers have developed a hybrid localization method that first use network technologies to compute in a few seconds an approximate position that is afterward used by the GPS device to improve the position accuracy (5 – 50 m). The Nokia N95 implements this functionality called Assisted GPS.

## 3) Map

---

An option when informing the user about the route to follow is to display it on a map. Hence, it seems relevant to study the different map applications available on the market. Thus, we will first discuss the pros and the cons of the different map providers. Then we will finally explain the problems we could face.

### 3.1) Nokia Map

It is logical to present you first the map application developed by Nokia since we are currently working with a Nokia N95. The last version supported by the S60 Nokia mobile phones (the operating system running on certain Nokia mobile phones) is Nokia Maps 2.0, and it offers multiple possibilities such as:

- The drive navigation – it delivers you to your destination thanks to a turn-by-turn visual and voice guidance
- The pedestrian navigation – it helps you to reach your destination going by foot
- The satellite views – They give you a new perspective wherever you are in the city



Let's now have a look at the graphical interface of the application. The picture below shows what you display on the screen if you wish to browse the map.

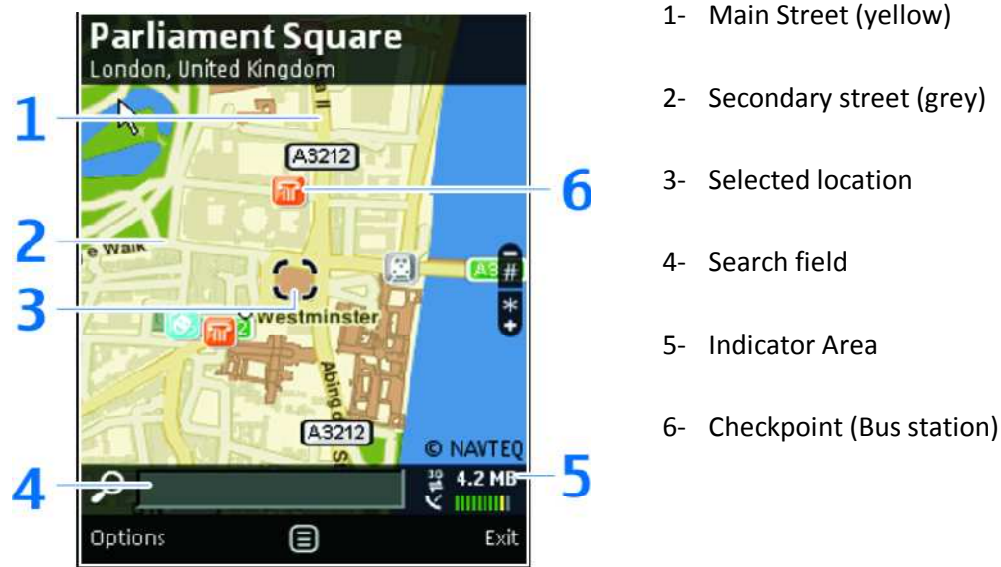


Figure 9 - Nokia map GUI example 1

The picture below shows what you display on the screen if you wish to navigate.

- 1- Direction (Big arrow)
- 2- Route (dark grey)
- 3- Your location
- 4- Information bar (speed, distance, time)



Figure 10 - Nokia map GUI example 2

Nevertheless, after few researches, it turns out that Nokia doesn't give a free access to the map API. Thus we unfortunately won't be able to explore that solution later for our project.

### 3.2) Navteq

Navteq is known for having one of the most comprehensive and accurate geographic database in the world. Moreover, Navteq now offers coverage in 73 countries on 6 continents. The company is now well settled over four key markets:

- Vehicle navigations systems
- Internet/Wireless
- Fleet/Entreprise/GIS
- Government

Indeed Navteq digital map drives most of the in-vehicle navigation systems sold in the U.S. and Europe. Plus, Nokia currently works with Navteq data coordinates for all its mobile phones (Observe the copyright on the Figure 9 - Nokia map GUI example 1). Thus it could appear to be a very good solution.

About the Navteq data coordinates, they are retrieved in a XML file, and respect this format:

“degrees, minutes, seconds.decimal seconds”, (e.g 47 degrees 25 minutes 25 seconds)

However the Navteq license is expensive and will require both money and time before being obtained. Since the technical part of the project needs to be done within six month, we should consider other open sources solutions which would suit better to that kind of project.

### 3.3) OpenStreetMap

[xi - Steve Coast, 2004] OpenStreetMap is an open source project created in July 2004 by Steve Coast. This project could be compared to a Wikipedia for map coordinates. Indeed everybody is allowed to add his own data or to retrieve OpenStreetMap's data for personal usage. Thus, even if OpenStreetMap doesn't provide the most relevant coordinate data, it remains a good compromise as data are already available for Aalborg city. Plus it is possible and easy for us to add manually the bus stations coordinate in the website database.

Even though it seems easy to retrieve data coordinate, it unfortunately is not as easy to retrieve the map view. Indeed, OpenStreetMap isn't optimized for mobile phone applications and retrieving a map picture requires the user to press a button. Thus, it appears to be a very odd solution since data coordinate are easy to obtain whereas map pictures are technically complicated to retrieve.

### 3.4) Google Map

Google is one of the leaders about map applications for mobile phone. Here is a short summary of Google map for phone's features:

- My location: it displays your current location on the map so that you can find where you are
- Driving directions – It shows you the way to your destination thanks to a turn-by-turn driving directions. Combined with the 'my location' feature, it helps you to find your way without typing your starting point
- Maps and satellite views - They allow you to switch from a map to a satellite view whenever you want

Basically, the options offered by Google map are pretty much the same as Nokia map 2.0 ones. The main difference lies in the graphical interface:



Google map mobile has tried to keep a graphical interface as similar as possible as Google map for computers. As you can easily see, it looks a bit like Nokia map 2.0 anyway.

Nevertheless, compare to OpenStreetMap, it works the other way around since it enables the user to retrieve data coordinates, but easily allows him to display a map picture.

Therefore, there are a lot of different offers on the market and none of them seem to be perfectly complete for our project, for technical or financial reasons. However several of them individually offer very interesting possibilities. Thus, the final solution will probably be the result of a combination between these ones.

## 4) Shortest Path Algorithms

---

Shortest path problem is inevitable in road network applications. Since many parameters evolve during a trip (bus delayed, user running out late, etc.), a huge amount of request occur at any moment and it needs to quickly find the solution. Thus, we are going to study three approaches: Dijkstra's shortest path algorithm, Restricted search algorithm, and finally A\* algorithm.

### 4.1) Dijkstra's shortest path algorithm

Conceived by Dutch Computer scientist Edsger Dijkstra in 1959, this algorithm solves the single-source shortest path problem for a graph. For a graph  $G(V, E)$ , where  $V$  is the set of vertices and  $E$  is the set of edges, the running time for finding a path between two vertices varies when different data structure are used. Dijkstra's algorithm helps to solve the problem of the shortest path between two edges in a connected graph. Let's study the algorithm below [xii]:

```
For each  $u \in G$ :  
     $d[u] = \text{infinity}$ ;  
     $\text{parent}[u] = \text{NIL}$ ;  
End for  
 $d[s] = 0$ ; //  $s$  is the start point  
 $H = \{s\}$ ; // the heap  
while  $\text{NotEmpty}(H)$  and  $\text{targetNotFound}$ :  
     $u = \text{Extract\_Min}(H)$ ;  
    label  $u$  as examined;  
    for each  $v$  adjacent to  $u$ :  
        if  $d[v] > d[u] + w[u, v]$ :  
             $d[v] = d[u] + w[u, v]$ ;  
     $\text{parent}[v] = u$ ;  
     $\text{DecreaseKey}[v, H]$ ;
```

As the number of nodes in a graph increases, the running time of the applied algorithm will become longer and longer. Indeed, the execution time of the algorithm doesn't increase linearly but exponentially. If the road network of a city has more than  $10^4$  nodes, a fast shortest path algorithm becomes more desirable. [xiii - Thomas Willhalm, 2005]

### 4.2) Restricted Search Algorithm

When the Dijkstra algorithm is used to find the shortest path, it starts search from the start point and spreads as a circle until the radius arrives the destination. Instead of search the entire circle, the Restricted Search Method only search with the small area of the remaining part of rectangle Rec2 cutting off by the two bold straight lines. The rectangle Rec1 has the straight line of  $S$  and  $D$  as a diagonal and Rec2 is a rectangle extended from Rec1 by a threshold  $T_2$ .

The two straight bold lines parallel the Instead of search the entire circle, the Restricted Search Method only search with the straight line of  $SD$  with a distance of  $T_1$ .  $T_1$  and  $T_2$  are two variables

that need to be decided to ensure that the optimal path is included within the restricted area. Depending on the situation (what we are looking for, where we are), the usual range is chosen to go from 500m to 1500 m.

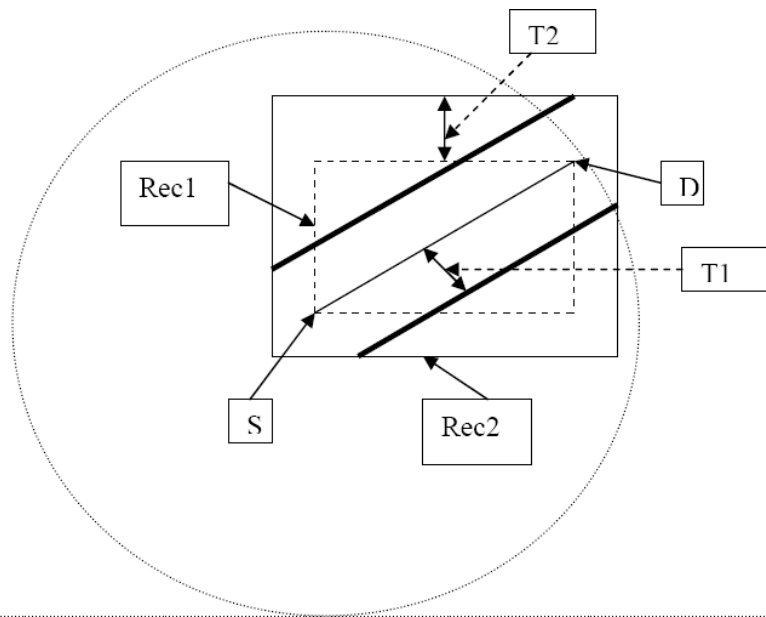


Figure 11 - Search Area of Dijkstra and Restricted Search Algorithm

```

for each u G:
    d[u] = infinity;
    parent[u] = NIL;
End for
d[s] = 0;
H = {s};
while NotEmpty(H) and targetNotFound:
    u = Extract_Nin(H);
    label u as examined;
    for each v adjacent to u:
        if outOfRange(v), then continue;
        if d[v] > d[u] + w[u, v], then
            d[v] = d[u] + w[u, v];
            parent[v] = u;
            DecreaseKey[v, H];
  
```

```

Procedure outOfRange(Constraint Area A, Vertex v):
  //A is a polygon given;
  //v is a Vertex being checked;
  Make a straight-line L from v to the right of v;
  Counter = 0;
  For each edge e of A
    if L intersects with e
      increase Counter by one;
  if Counter is even
    return true;
  else
    return false;

```

The checking procedure is implemented by counting the number of intersection of the horizontal line starting from the node to the right with the restricted area. The following figure illustrates this method:

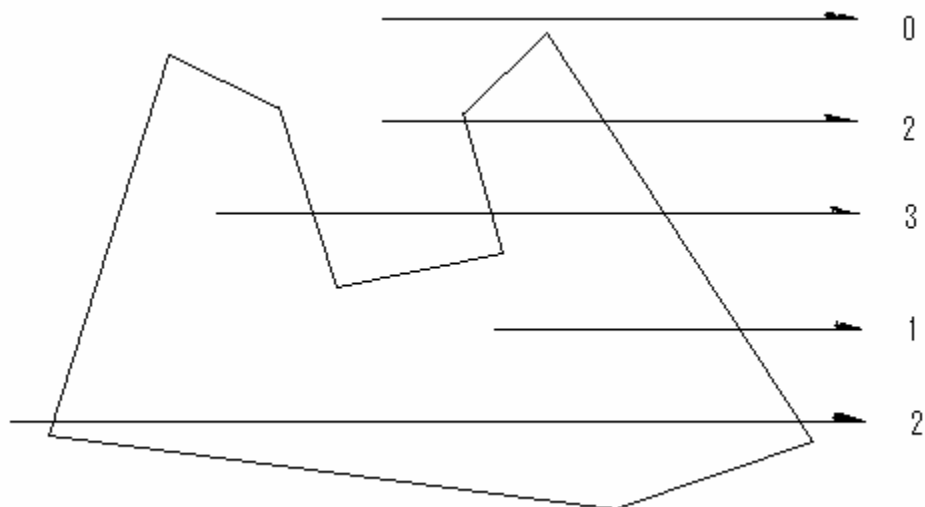


Figure 12 - Check point in polygon

Nevertheless, two problems remain with this method. First of all, the two thresholds won't get the solution properly since some high ways may be beyond the restricted area and not being taken into account even though the shortest path should normally go through them. Plus, as the distance between the two points increases, the shortest path more likely spreads wider from the straight line of SD.

### 4.3) A\* search

The A\* search integrates a heuristic into a search procedure. Indeed, for each vertice, it estimates the proximity to the final destination. This method uses Euclidian distance as estimated distance to the destination as shown below:

$$f(V) = \text{distance from } S \text{ to } V + \text{estimate of the distance to } D^*$$

\* With S the start point, V the vertices, and D the destination

$$f(V) = d(V) + h(V, D)$$

$$f(V) = d(V) + \sqrt{(x(V) - x(D))^2 + (y(V) - y(D))^2}$$

Where  $x(V)$ ,  $y(V)$  and  $x(D)$ ,  $y(D)$  are the coordinates for node V and the destination node D.

*for each u G:*

*d[u] = infinity;*

*parent[u] = NIL;*

*End for*

*d[s] = 0;*

*f(V) = 0;*

*H = {s};*

*while NotEmpty(H) and targetNotFound:*

*u = Extract\_Min(H);*

*label u as examined;*

*for each v adjacent to u:*

*if d[v] > d[u] + w[u, v], then*

*d[v] = d[u] + w[u, v];*

*p[v] = u;*

*f(v) = d[v] + h(v, D);*

*DecreaseKey[v, H];*

The A\* search algorithm is:

Since the Aalborg is quite a big city, so it has a lot of intersections, a tree representing it contains approximately more than 10000 nodes. Hence, the Dijkstra's algorithm remains the most appropriate when treating over  $10^4$  nodes (cf. 4.1). Therefore, it remains the most accurate algorithm, and Aalborg city's size allows easily Dijkstra's calculations

## 5) Application Server

---

As it is said previously, a mobile phone is not powerful enough to compute a big amount of calculations. Therefore, we might need to use a server to compute them instead. Thus, it is necessary to choose between the different operating systems currently available on the market.

Instead of focusing on the user interface, server operating systems are made to be secure, stable and to provide distributed applications. Nowadays, there are two main operating systems used for servers: Unix-core based and Windows-core based. We will talk about both of them.

### 5.1) Unix-core based operating system

Nowadays, there are a great number of UNIX systems, and an even bigger number of distributions.

Of course, the main characteristic of Linux distributions is the fact that they are open-source; it means that they are without any cost and also the code is available to everyone. Besides that, nowadays we can install Windows programs on a Linux system, using emulators like *Wine* (running on an Intel X86's platform).

One of the disadvantages of Linux systems is that the user that will install the OS has to know how to work with *Linux Shell*, while people are more familiar with Windows. Otherwise, it will be very complicated to install some programs. Another problem of Linux lies in the broken packages: sometimes the user wants to install a package (program) in Linux, and he has to install a lot of dependencies first.

### 5.2) Windows-core based operating system

Windows is the most popular OS in the market. One of the main reasons is that it has a user-friendly interface and is much easier to use than Linux.

Concerning the server, the most recent in the market is the Windows Server 2003. But this edition is better to enterprises or companies that require a lot of computers connected to each other or need a lot of computers running the server application.

Besides the server edition, exists also another editions of Windows, like Windows XP and Windows Vista.

As Linux have its advantages, for Windows is precisely the opposite, is not Open-Source, neither is free. Actually, costs a lot to buy a Windows license (approximately 1000 USD).



### 5.3) Conclusion

For our application server to run, we will only need one computer, with an operating system able to run programs on the common programming languages, so we can choose the one we want, and allowing us to use different development environments.

At some point, during the process of providing the user with a route based on real bus times, we will have to retrieve this information from a system or another. Let's now see how we can do that.

## 6) Bus schedule

---

One objective of the system is to be as accurate as possible regarding the timing. Hence, we have to study the possibility to deal with the bus times. That is what we will do in the following. As our project is aimed to be used in Aalborg city, we will focus on Aalborg buses. These are run by a private company, called NT.

### 6.1) Fixed schedules

The first option available would be to store all the times of all buses. It means more than 20 buses, with for each of them, their week days and week end times, for the night and the day, during the holidays, and so on...

The source of information could be the NT website [xiv - Nordjyllands trafikselskab, 2008]:

Lindholm Station	Vesterbrogade	Nytorv	Aalb. Busterminal A	Bornholmsgade	Grønlands Torv	AAU Kroghstræde	Universitetet (AAU Busterminal)
5.22	5.26	5.31	5.40	5.43	5.47	5.53	5.55
5.52	5.56	6.01	6.10	6.13	6.17	6.23	6.25
6.07	6.11	6.16	6.20				
6.07	6.11	6.16	6.25	6.28	6.32	6.38	6.40
6.24	6.27	6.31	6.35				
6.22	6.26	6.31	6.40	6.43	6.47	6.53	6.55
6.36	6.41	6.46	6.53	6.56	7.01	7.07	7.11
7-8 minutter							
8.36	8.41	8.46	8.53	8.56	9.01	9.07	9.11
8.43	8.48	8.53	9.00	9.03	9.08	9.14	9.16
10 minutter							
13.33	13.38	13.43	13.50	13.53	13.58	14.04	14.06
13.43	13.48	13.53	14.00	14.03	14.08	14.14	14.18
7-8 minutter							
17.58	18.03	18.08	18.15	18.18	18.23	18.29	18.33
18.14	18.17	18.21	18.30	18.33	18.37	18.43	18.45
		18.36	18.45	18.48	18.52	18.58	19.00
18.44	18.47	18.51	19.00	19.03	19.07	19.13	19.15
18.59	19.02	19.06	19.15	19.18	19.22	19.28	19.30
14	17	21	30	33	37	43	45
		36	45	48	52	58	00
44	47	51	00	03	07	13	15
59	02	06	15	18	22	28	30
22.14	22.17	22.21	22.30	22.33	22.37	22.43	22.45
		22.36	22.45	22.48	22.52	22.58	23.00
22.44	22.47	22.51	23.00	23.03	23.07	23.13	23.15
22.59	23.02	23.06	23.15	23.18	23.22	23.28	23.30

Hvis du skal til Storvorde, Klarup eller Gistrup se side 68 - 79.

**7-8** Bus hver 7-8. minut.

**10** Bus hver 10. minut

Figure 13 - Bus schedule example

To store all that, we could use:

- An XML file: technically easy to implement, but hard to interact with, and pretty heavy for the system.
- A database: quite long to implement, but easy to consult. Several famous provider offer database solutions, e.g. MySQL, PostgreSQL,...

## 6.2) Live schedules

NT already implements a service that displays, in real-time, the times of the buses. It is called "NT Live" [xv - NT Live, 2008].


  
**AAU Busterminal - 18:06**


---

54 mod Aalborg Busterminal  
... 18.02 +4

2B mod Nørhøne/Airport  
... 18.07

2L mod GISTRUP (Sæderupvej)  
... 18.11

12 mod Vesterkæret  
... 18.13

14 mod AAU Busterminal  
... 18.14

14 mod Skelagervej v/Skelag  
... 18.14 \*

12 mod AAU Busterminal  
... 18.14 \*

2H mod Klarup  
... 18.18 +1

36 mod AAU Busterminal  
... 18.24

2C mod Uttrup Nord  
... 18.26 \*

*\* = Køreplantid*  
**Se flere afgang**

---

Hovedmenu  
Om NT LIVE service  
Tilføj bogmærke  
Tip en ven

---

**mobil.ntlive.dk** 

Figure 14 - Live schedule example

On this website, the times are displayed with the delay applied to the time: “+1” means that the bus has been delayed by 1 minute.

So it only provides a time accuracy of one minute.

To sum up on the bus times, we don’t have so many ways to know the schedules. Whether we rely on the basic, planned, schedule, or we try to use the live service somehow. We will see later how we actually proceeded in our project.

As a conclusion of our analysis, we can say that we now have analyzed the different technologies, algorithms and devices we will have to deal with during our project. Regarding the analysis we did and the project constraints we have such as the short amount of time or money, we will discuss the different choices we will make and explain them through the project’s design.





# Design

In the previous part of this report, we have been focusing on the different parts and the different elements of the project. We will now study the project as only one entity, and we will explain how the different parts shown in the previous pages are interacting with each other.

## 1) Structure

---

### 1.1) Global Diagram

In order to do so, a global diagram will be displayed. This global diagram is a quite simple one, summarizing how the whole project works. Each element is shown as a square or another geometrical shape, and the interactions between those elements are described by arrows.

The purpose of this diagram is not to go into the details for each function of the product, but to offer a global view of the project.

See diagram on the next page.

### 1.1.1) Diagram

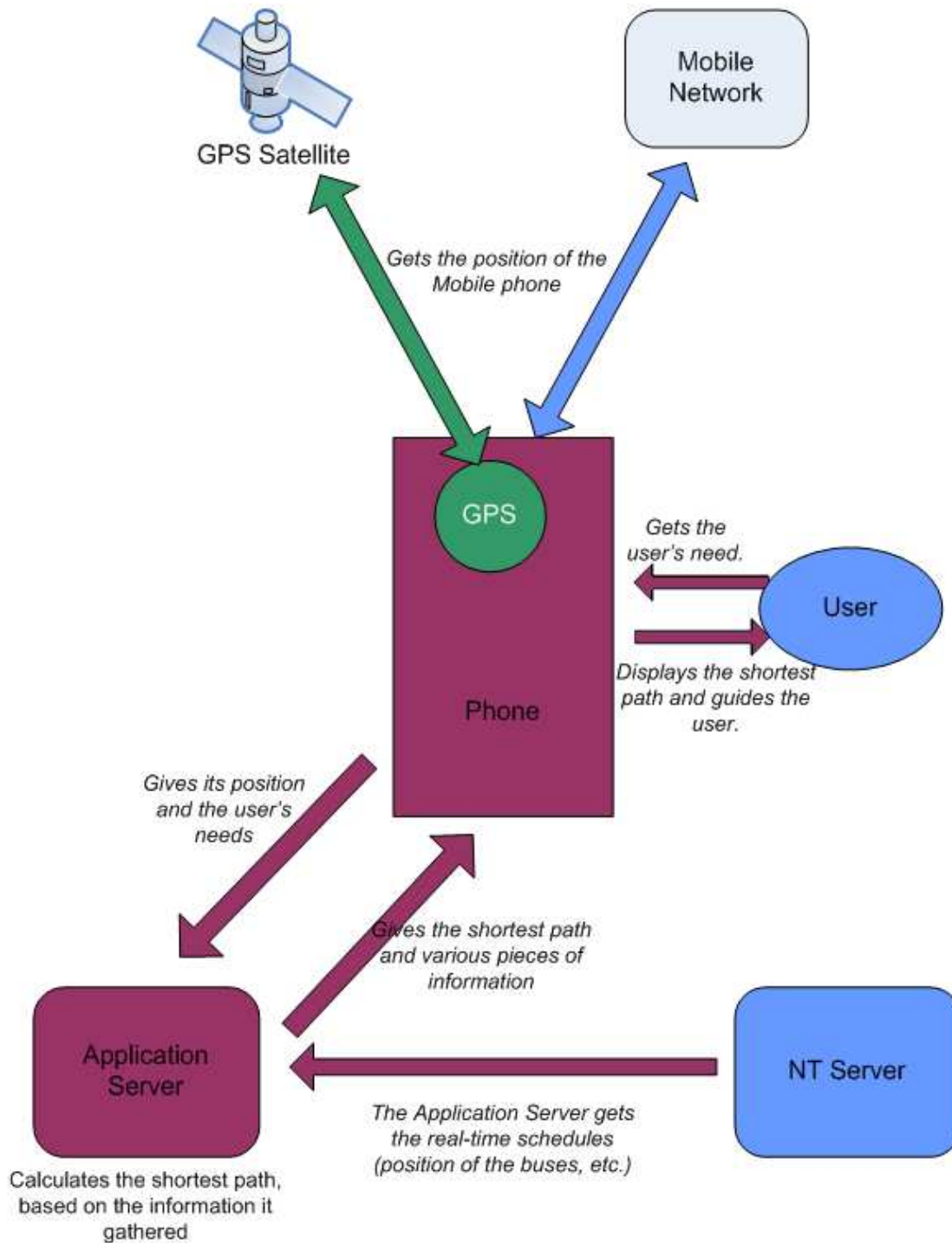


Figure 15: Global Diagram  
(elements in purple are the ones we will develop ourselves, elements in green are optional)

### ***1.1.2) Explanation of the diagram***

First, we will quickly explain the diagram itself. In the center of this diagram, we can notice the mobile phone. It is indeed a very important part of the product, since the mobile phone application will be installed on it. This mobile phone contains another entity: the GPS. The entity is not really described in the diagram, since it is already available in the N95 mobile phone. Nevertheless it might happen that some mobile phones don't have a GPS as an internal device, and an external one could be added easily.

This GPS communicates with the satellite, following the protocols of the GPS system. We will not go into the details of this communication, since it is not the purpose of our project. All we need to know is that the GPS can get the position of the mobile phone, and thus the position of the user. It also gives us the speed of the user and the accuracy.

The mobile phone uses this information to respond to the user's need. The user actually interacts with the mobile phone through the mobile phone application. In order to treat the user's demand, the mobile phone interacts with a remote server, that we will call the application server.

The application server interacts with both the phone and the NT server. The NT server, which belongs to the NT Company, contains all time table of the buses. In order to perform calculation properly, the application server needs to exchange data with the mobile phone (to know the user's destination, to get the route info, etc.) and the NT server (to know the schedules).

This is a view of the whole project, but we will actually only develop the application server and the mobile phone application parts. But, since those parts interact directly with the rest of the system, we need to study all the elements listed in the global diagram.

## **1.2) The GPS interactions**

The GPS is not a necessity, but it is recommended for the system to work with maximal capacities. If there are no GPS, then the application will rely on network positioning to get its position.

## **1.3) The Mobile Phone interactions**

### ***1.3.1) The Mobile Phone***

The mobile phone is the only part that truly interacts with the user. Indeed, the user will not see the application server and the NT server. Neither will he directly interact with the GPS. As a matter of fact, the application on the mobile phone must be well designed and user-friendly. The user must be able to clearly state his need, and the answer to this need should be shown clearly on the mobile phone. In our case, the user's input will be, most of the time, the address of the place where he wants to go, and the output will be the instructions to follow.

### ***1.3.2) Interaction Mobile Phone / GPS***

In order to operate efficiently, the system needs to know the position of the user. This information will be given by the GPS. Since this piece of information is needed to calculate the path, the application of the mobile phone will have to interact with the GPS to retrieve its position.

This means that the application must communicate with the GPS. This is done with the API of the GPS, which is embedded in the mobile phone. The API uses java technology, which enables the mobile phone application to easily listen to the GPS data stream. From this data stream, the application can deduce the position of the user, as well as many other useful pieces of information, such as its speed, the direction he is going towards, etc. All this information will be used to calculate the shortest path, and to guide the user during the trip.

### ***1.3.3) Interaction Mobile Phone / Application Server***

As we have seen, the mobile phone gets access to some information through the GPS and the user's input. This information is needed to determine the shortest path, and to guide the user. However, this calculation is not done by the mobile phone application. Most of the calculation will be done by the application server, which is more powerful, and which gets information about the buses schedule.

That is why, the user's destination, his position, and a few other parameters will be sent from the mobile phone to the application server. With this information, the application server will be able to compute a result.

The connection between the device and the server will be performed thanks to the internet. The device indeed needs to be connected to the internet to use the application.

## **1.4) The Application Server Interaction**

### ***1.4.1) The application server***

The whole system is supposed to treat a lot of information: the bus schedules, the position of all the users, the paths, the destinations, etc. There is also a lot of information that is not related to a specific user. That is why, the system needs a server to compute the information and calculate the paths. Plus, the server offers us a more important computing power. Therefore, it increases the general speed of the mobile application. Thus, we chose to have an application server to communicate with all the users' devices.

### ***1.4.2) Interface Application Server / Mobile Phone***

As we have seen, the application server retrieves the user's requests, and calculates the paths, or other information required. It must then send back this result to the mobile phones, which will display it. Actually, the server will be exchanging information with several devices. If a mobile phone loses its connection to the server, it will not receive any guidance.



### ***1.4.3) Interface Application Server / NT Server***

To establish a path from the starting point to the destination point, the server needs information about the buses: schedules, position of the buses, and miscellaneous information. This information is available on the server of the NT Company. That is why; our application server will have to connect with the NT server, and retrieve this information.

### **1.5) The NT Server**

The NT Server is not really part of the system, because it belongs to NT. Thus, the application server will communicate with it so that its data will be available to our system.

## 2) Mobile phone application

---

First of all, we studied some differences we could see between different mobile phones: size of the screen, compatibility of specific functions, etc. As we figured that it would be extremely difficult to build an application compatible with different devices. Moreover, the duration of the project didn't allow us to try to solve these issues. Hence, we only built and tried our program on the mobile phone we had: the Nokia N95. However, as several Nokia phones run the same operating system (Symbian: S60 version), the program should work, and the problem would only be graphical because of the different size of the screens.

### 2.1) Destination Mode

The most important feature of the mobile phone application is to guide the user to its destination. That is the main purpose of our project, so it was an important part to design.

We decided there would be two important objectives in this mode. The first objective was to find a way to display the path on the screen of the mobile phone, to show the user the shortest path to his destination, and which buses he has to take. The second objective was to guide the user to his destination. We will thus study how we designed those two parts separately.

#### 2.1.1) *Display the Path*

To find and display the path on the screen, there are several steps that we had to consider. First, we needed to know where the user was, and where he was headed. On another hand, we needed to find a good way to show the result given by the server.

##### 2.1.1.1) The information needed

The first element we have to consider is the fact that we need to get a starting point, as well as an ending point. The destination point must be given by the user. We decided that the user would give an address for the destination. As we needed to get the user's position as well, we thought about several solutions:

- If a GPS is available, then we can retrieve the GPS coordinates of the user.
- If no GPS is available, then we have to ask the user for his current position. Since it is highly improbable that the user knows his GPS coordinates, the system would rather ask him for his current address.

When the system has retrieved that information, it sends them to the server, which will make the shortest path calculation.

##### 2.1.1.2) Display the path on a map

When the server is done with the calculation, it sends the path data to the mobile phone. The role of the mobile application is then to show the user this path. Several ways to do this were available to us. We wanted to get the same kind of features as a real GPS device for cars. That is why

we thought it would be interesting to display the path on a kind of map. However, we decided that, unlike the GPS devices for cars, it would be a simple map of the surroundings, where the path would be displayed. We thought it would be a very good way to tell clearly to the user where to go. Indeed, most people are used to read maps, so this would not be hard for the user.

We decided to use the Google Maps API to get the map. It was the best free solution that we got. As said in the “Analysis” part of this report, we listed every map APIs available to us, and looked for their advantages and drawbacks. To get a picture of the map, Google Maps API is the easiest solution, since it is possible to do it with a HTTP query.

In order to follow the user’s position, the map is refreshed when the user moves. This part is more documented in the “User Interface” part of this report.

#### 2.1.1.3) User guidance

- *Localize the user on the path*

We have now retrieved both the user’s position on the map and the path. Separately, these two pieces of information aren’t useful, but computed together they offered us a very interesting view of the user’s moves along the path. Since we achieve successfully to calculate the user’s position on the path after many attempts, we will first explain the methods we went through, the problems we faced, how we solved them, and finally we will explain the solution we kept.

*First method: The distance from the user to a node*

At the very beginning we programmed a very easy method to calculate the distance from the user to the path. Indeed, we just determined the distance from the user to the path as being the shortest distance from the user to a node of the path. The sketch below shows how the method works:



Figure 16 - user to point distance

Algorithm:

```
/* Variables declaration and initialization */
```

```
The shortest distance = Maximum variable type value
```

```
The closest node = Random value
```

```
For each node "i":
```

```
    distance =  $\sqrt{(x_{user} - x_{nodei})^2 + (y_{user} - y_{nodei})^2}$ 
```

```
    If (distance < the shortest distance), then:
```

```
        The shortest distance = distance
```

```
        The closest node = index "i"
```

```
    End if
```

```
End for
```

For instance, if we apply the algorithm to the example previously drawn, we will have d4 as the shortest distance, and consequently 4 as the nearest node.

Nevertheless, after many trials it turns out that the method wasn't really accurate for many different reasons. We will list and explain all the different problems we faced:

First of all, if the path is made of two really distant nodes, it might happen that the user is detected as being close to a point which is not defining the street he currently walking down. The following picture illustrates this case:



Figure 17 - First problem faced

The user should logically be localized between the nodes b and c, but since the part of Danemarksgade he is currently walking through is just defined by two really distant nodes (b and c), the shortest distance between the user and all the path's nodes isn't  $d(c)$  nor  $d(b)$  but  $d(a)$ .

Besides the localization problem on the path, this method doesn't allow us to know with a good accuracy where the user is in the street. Indeed, we just know which point is the closest from the user. Thus, we decided to look for other solutions to solve that localization problem.

#### *The user to straight line distance*

That method is the best we found to estimate the position of the user on the straight lines defined by the nodes. Contrarily to the previous method we used, the current one requires some previous calculations. We will first explain these previous calculations we did, and then we will develop how we obtained the distance from a straight line to a point. Finally we will explicate few tips we used to make the program a bit faster.

- Pre-requires

If we want to calculate the distance from the user to a straight line, we obviously need both the user's position, and the straight line equation. Thus we will explain in the short coming algorithm the equation Object we created in the program, as well as the methods which define it.

A linear equation is defined by:  $y = ax + b$ , where 'a' is the slope, and b a constant. Plus, we need to define the interval between the two nodes. Let's write the interval as being [Xmin, Xmax]. Here is the object's constructor algorithm:

#### Algorithm:

Constructor parameters: Coordinate of the first node and coordinate of the second node

/\* Calculation of the slope and the 'b' constant \*/

$$a = \frac{(\text{latitude}_{\text{second}} - \text{latitude}_{\text{first}})}{(\text{longitude}_{\text{second}} - \text{longitude}_{\text{first}})}$$

$$b = \text{latitude}_{\text{first}} - a \cdot \text{longitude}_{\text{first}}$$

/\* Calculation of the interval [Xmin, Xmax] \*/

If second node's longitude is greater than first node's one, then:

$$X_{\min} = \text{longitude}_{\text{first}}$$

$$X_{\max} = \text{longitude}_{\text{second}}$$

End if

Else

$$X_{\min} = \text{longitude}_{\text{second}}$$

$$X_{\max} = \text{longitude}_{\text{first}}$$

End else

At that point, we just have the equations linking consecutive nodes together. We now need to calculate the distance from the user to each segment. As everyone knows, the distance from a point to a straight line is the perpendicular distance from that point to that line.

- Calculation of the distance

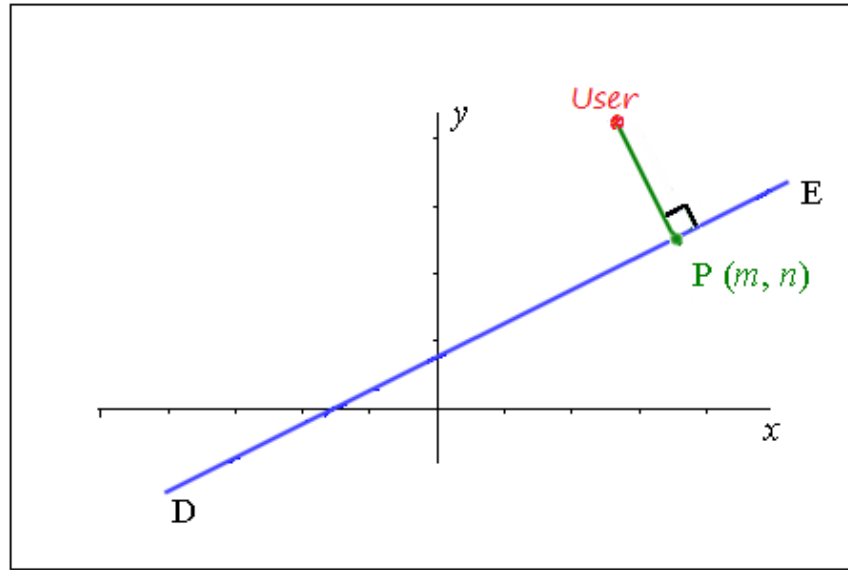


Figure 18 – Modify from intmath.com

Knowing the user coordinate and the (DE) equation, we now can calculate the equation of the straight line (PU). Let's say that (DE):  $y = ax + b$ , and (PU):  $y = a'x + b'$ . Since these two equations are perpendicular, we will have  $a' = -a$ . Thus, we are able to calculate  $b'$  value:

$$\begin{aligned} y_u &= a'x_u + b' \\ b' &= y_u - a'x_u \\ b' &= y_u + ax_u \end{aligned}$$

Since P is both on the (PU) and (DE) straight lines, we can solve the following equation:

$$\begin{aligned} L1: \quad & \begin{cases} y_p = ax_p + b \\ L2: \quad y_p = a'x_p + b' \end{cases} & \Leftrightarrow & L3 = L2 - L1: \quad \begin{cases} y_p = ax_p + b \\ y_p - y_p = a'x_p - ax_p + b' - b \end{cases} \\ L1: \quad & \begin{cases} y_p = ax_p + b \\ L3: \quad 0 = a'x_p - ax_p + b' - b \end{cases} & \Leftrightarrow & L1: \quad \begin{cases} y_p = ax_p + b \\ L3: \quad 0 = -ax_p - ax_p + b' - b \end{cases} \\ L1: \quad & \begin{cases} y_p = ax_p + b \\ L3: \quad 2ax_p = b' - b \end{cases} & \Leftrightarrow & L1: \quad \begin{cases} y_p = ax_p + b \\ L3: \quad ax_p = \frac{b' - b}{2} \end{cases} \end{aligned}$$

And finally:  $y_p = a \cdot \left( \frac{b' - b}{2} \right) + b'$

We now have obtained the coordinate of the point P, and before we start any distance calculation from the user position to that point, we should first check if P is located between the two nodes. So basically, if  $x_p \in [Xmin, Xmax]$ , then:

$$distance = \sqrt{(x_u - x_p)^2 + (y_u - y_p)^2}$$

However, after many trials it turns out that the method we used was not perfect. Indeed, in very few cases the user was considered as lost. It took us a little while to figure out why we obtained such results. In order to make it clear, let's first have a look to the following figure:



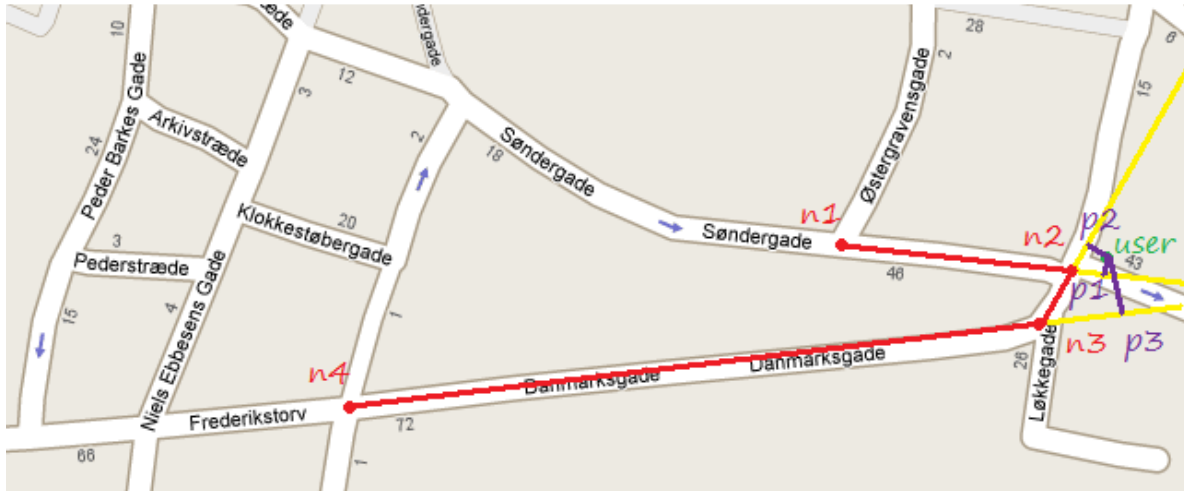
Figure 19 - Second problem faced

The above example represents a really common practical case. Indeed, 4 nodes define the path, and the user is located few meters away from the path since the GPS never is perfectly accurate. To make it easier to understand, we drawn the straight lines linking the nodes together:

- The red line represent the part of the equation inside the interval  $[Xmin, Xmax]$
- The yellow line represent the part of the equation out of  $[Xmin, Xmax]$  which interests us for the case study.

As you can easily observe, none of the  $P_{i\{1, 2, 3\}}$  points drawing a perpendicular straight line from the user position to the 3 different linear equations are located on the yellow part of the lines. Basically it means that no distance will be calculated with the previous method explained, and the user will be considered as lost.





In view of the fact that we have already worked on another distance calculation method, we thought that handling that special case exception by using the previous method would be a benefit since we already had a code. Therefore as we have that special case, we assume that the distance from the user to the path is the distance to the closest node. The combination of these two methods seems to work pretty well since they both prevent each other from their inherent problems.

Therefore, we can now read into this information to interact with the user.

- *Trigger the actions*

Once we know where the user is located on the path, we can interact with him. But first of all we have to define some checkpoints, calculate the distance from the user to these checkpoints, and finally trigger the relevant event. Thus, we will present you these steps in the same order.

#### *The checkpoints*

As you already know, a path is defined by steps, segments and nodes. As a really short reminder, each step is made of segments and each segment is made of nodes. Steps define if the bus user is in a bus or not, and segments are defining streets the user will have to walk through. Consequently, the checkpoints will always be:

- The end of a segment for a change of direction (turn right/left, walk straight ahead messages)
- The end of a step for a means of transportation's change (get in/off the bus messages), or the end of the path (arrival message)

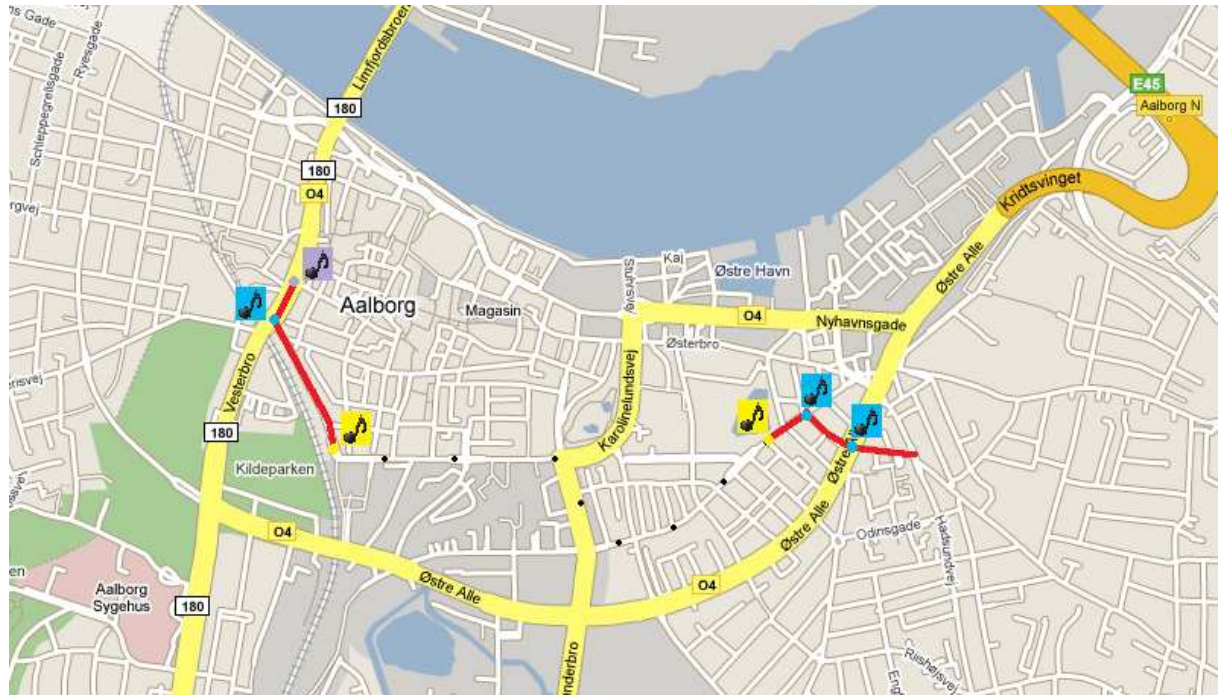


Figure 20 - Checkpoints

On the map, blue notes represent a change of direction, the yellow ones a means of transportation's change, and finally the purple one represents the arrival point. For each notes, we will play a different sound indicating the user the direction he should follow, the bus he should get in, when he should get off, and when he is arrived.

Now that we have defined the checkpoints, we will explain how we calculate the distance from the user to them.

#### *The distance from the user to the checkpoints*

- To the end of a segment

Once we have calculated the distance from the user to the path, then we have to add the distance from the P point to the end of the segment. The distance calculation between two points is really easy; nevertheless we still have to define the right interval to work on. In order to make it easier to understand, let's first have a look at the algorithm:

```
/* Distance from the user to the path has been previously calculated and stored in the tempDist variable */
```

```
// We first calculate the distance from the P point to the end of the current straight line
```

```
tempDist = tempDist + distance from P to equation index + 1
```

```
For i = shortest equation index + 1 To the size of the segment:
```

```
tempDist = tempDist + distance from nodei to nodei+1
```

```
End If
```

The equation list, as all the other lists, starts from the index zero. Thus, the first distance we will have to calculate will always be from the “shortest equation index +1”. Let’s clear up this point with help of a schema.



Figure 21 - distance to the end of the segment

If we apply the algorithm to this case, which means that we already have calculated the distance from the user to the P point, we will first need to calculate the distance from the P point to the next node. As it is written in the algorithm, the next node has the index: equation number +1. Indeed, the first calculation only concerns the distance from P to the node 1 (index 0 + 1).

And finally in the “for loop”, we consecutively add the distances between each nodes belonging to the segment.

- To the end of a step

Actually the distance to the end of a step is quiet similar. Instead of just performing the calculation for a segment, we do it from segment to segment until we reach the end of the step.

Now that we have studied how to calculate the distance from the user to a checkpoint, we will explain how we chose to trigger the relevant events.

*The actions*

- Change of direction

The most efficient way to properly explain that part of the project is certainly to add a small piece of the program to the report. Without a doubt, it will be pretty easy to understand since it is the last step, and you already know all the variables we have previously calculated. Basically, if the user is located below a certain distance (established as 50 meters in most cases) to a checkpoint we will play a sound related to an action which has already been defined in a table. Once a sound has been played, a boolean variable which has been named explicitly will be set to "true". This value will be set back to "false" as soon as the checkpoint will be reached. Thus, the program will stay focused on the next checkpoint coming in the path. Let's now have a look at something more technical:

- Means of transportation change

The data we need to be aware of a change of means' transportation are exactly the same. However, the conditions which have to be respected to trigger an event are quite different. Indeed we created two cases:

1. The user arrives to the bus station and he has to get in a certain bus of a certain line
2. The user has to get off the bus

For both cases we need to calculate the distance from the user to the end of the step. However, we won't play the sound at the exact same position depending on the case. Indeed, in the first case, that distance will have to be below 10 meters. After few trials it turns out that the sound was played a bit too early if we handled this event as we did for the change of directions.

```
if(tempDist < 50 && soundAlreadyPlayed == false
&& shortestSeg <= Main.screenMap.path.listSegments.size()-1)
{
    Switch (listSegments.elementAt(shortestSeg+1)).action)
    {
        case 0 :
            Main.sound.setSound("ahead");
            soundAlreadyPlayed = true;
            break;
        case 90 :
            Main.sound.setSound("right");
            soundAlreadyPlayed = true;
            break;
        case 270 :
            Main.sound.setSound("left");
            soundAlreadyPlayed = true;
            break;
    }
}
```

The way we handled the second action is very different. Instead of triggering the event based on a distance calculation, we will trigger it only if the user has reached the last but one bus station. We figured out that it was a much better solution since a sound played few meters before the arrival bus station could be a bit confusing. Plus, the signal strength in the bus might sometimes be a bit low and could delay the start of the sound. If we assume the worst case, the sound could even be played after the user passed the station. The tests we performed reveal that the last but one bus station method was the best one and would prevent from errors. Plus it lets some time to the user to press the bus button to indicate he wants to get off the bus at the next stop.

- Arrival point

At last but not least we have the arrival event. Indeed, a sound is played as soon as the user has reached the final destination, and not 50 or 10 meters before. Then we created a method which disables the mobile phone to play any sound as the user is arrived.

Even though this application offers a lot of advantages, the user could sometimes just need a simple view of the city's map. Thus, we will present you how it works in the next part.

## **2.2) Map mode**

Since the beginning, we planned not only to do the destination mode, that is our project aim, but also to make our program more complete, provide such a map mode, without any route displayed on it.

Thus, the main objective of this part of the mobile application is to provide to the user a way to see the map, with a simple interface. Although that, it is necessary to have the localization of the user, that it means that the mobile has to receive GPS signal while the user is using the map mode.

The map mode also includes Zoom In and Zoom Out functions. With this the user can explore a region that he is more interested, or simply view a bigger area.

Concluding what we said, with this function the user will be able to see the map without entering any address, as before. For that reason, we decided to include in our program, since it is an important extra for a router application.

## **2.3) Favorites and Settings**

### ***2.3.1) Favorites***

To make our application more user-friendly, we decided to add a favorite system. A favorite is an address the user has saved. The favorite has a name which is a sort of identifier. The purpose for the user is to avoid writing the address each time he wants to use it. With the favorite, the user can ask the system to go to this address, without typing the address again.

#### **2.3.1.1) Elements contained by a favorite**

We decided to keep the structure of a favorite very simple. Since it is supposed to keep an address in memory, it should contain the street, number, city and country. Moreover, the user needs to be able to associate a name to this address: for example, "home". This name is also used as an identifier of the favorite.

#### **2.3.1.2) Favorite Management**

We thought that the user should be able of course to manage his own list of favorites. The application should handle the editing of a favorite, the creation and the deletion.

### ***2.3.2) Settings***

As any application, it seemed normal to define some parameters that would be accessible and modifiable by the user. This let the user change the application so that it fits him in a better way.

### 2.3.2.1) Sound

The “Sound” parameter can only take two values: ON or OFF. If the sound is ON, then a voice will guide the user to his destination. If the sound is deactivated, then the user will not be guided by the voice.

### 2.3.2.2) Language

The application can be translated to another language. All the labels or any kind of indication on the screen will be translated into the selected language. The application is available in English, Danish, French, and Portuguese.

### 2.3.2.3) Path preferences

It seemed interesting to let the user give some preferences for the path calculation. For example if the user does not want to walk too much, he should be able to tell the application and the application should be able to handle it. The path preferences are “The less walk”, “The less connections” and “The fastest way”.

## 2.4) User interface

### *2.4.1) The steps to build the interface*

A user-friendly and well-designed interface is as important as the efficiency of the program algorithms. For that reason, we have spent a long time discussing various ways to obtain the user’s need and to present the information on a small screen. We first agreed on some basic principles, like the disposition of the menu item, and then we drew a first draft. A Lo-Fi prototype is designed based on this first draft in order to perform a few tests. Then, from the results obtained, modifications are applied. Finally a Hi-fi prototype is designed in flash in regards of the multiple corrections we did. Thus, we will describe how we went through these steps in that exact same order.



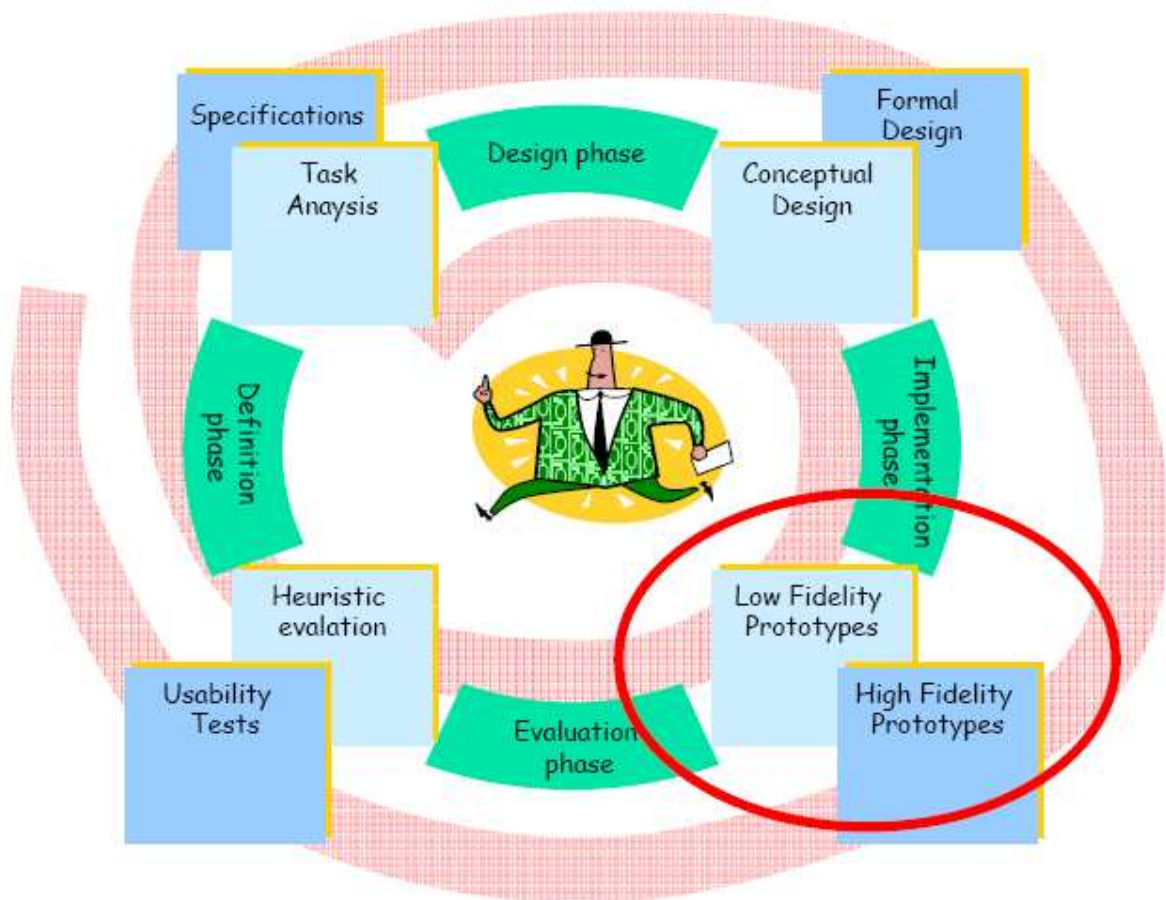


Figure 22 - Design spiral model

## 2.4.2) Main ideas of the interface

### 2.4.2.1) User's profile

In order to build the application, we had to identify clearly the type of users that will use our application. Indeed, depending on the personalities and the motivations of the people who use our product, the purpose of the interface will vary. In our case, the user will be both a bus and mobile (smart) phone user who wants to know quickly how to go from one point to another.

### 2.4.2.2) A simple interface to get the user's need

That is why; we thought that the user wouldn't like to interact with a complex application. The interface should be user-friendly and simple, in order to deliver quickly the information the user requires. It was our opinion that the user wouldn't want a complex and exhaustive interface, with many parameters to check before getting a result. He would rather want to enter only his destination and get immediately the description of the path to take.



#### 2.4.2.3) A graphical display to show the path

For this description of the path, we decided that a graphical display would be better suited. Indeed, nowadays all the GPS applications integrate a graphical display on a map, because it is the best way to ensure that the user visualizes clearly the way he has to take. It was thus really important that our application displayed the path in a similar way. Nevertheless we have to keep in mind that a mobile phone screen is much littler than a car navigation system. Thus, the overview should only contain the principal information.

Moreover, the user would be able to zoom in and out, and thus get a global view of the whole trip, or focus on some specific part of it. We preferred to make the application flexible and adapted to the user, letting him manipulate the map display as he wishes. Again, we decided to keep it simple, with not a lot of textual information on the map, apart from the arrival time.

#### 2.4.2.4) The textual mode

Displaying the information on a map has a lot of advantages, but it has also some important drawbacks, like for example the lack of text information, which can make the street name hard to guess, for example. For this reason, we decided to add an alternative mode to display the information of the path. The user would be able to switch from one mode to another quite easily, both modes displaying the same kind of information but in a different way. The text mode would present the path step by step. Each step will be represented by an icon and some instructions (for example: "turn left").

#### 2.4.2.5) The favorites

Even if the application had to be simple, some features like the favorites seemed really impossible to skip. To be able to memorize an address can really be convenient, and save a lot of time to the user. As a matter of fact, the favorite system is a quite simple way to save time. Moreover, most people are now used to favorites, because they use them in all their applications (e.g.: browsers).

#### 2.4.2.6) Settings of the application

As stated before, we decided to put very few settings in the application, because it seems pointless to offer the user a long list of parameters that he would only rarely use. Also, we decided to gather all the settings together in this one screen. Indeed, we could have chosen to put the user's preferences for the path in the destination menu, but this would have split the settings information in two different groups, causing confusion for the user.

#### 2.4.2.7) Possibility to read the map

We also thought that sometimes the user may want to read the map of the surroundings, without typing a destination, just to study its position. We thus decided to add a Map menu, which would display a map with the user position as a center, but no path.

### 2.4.3) LoFi Prototype

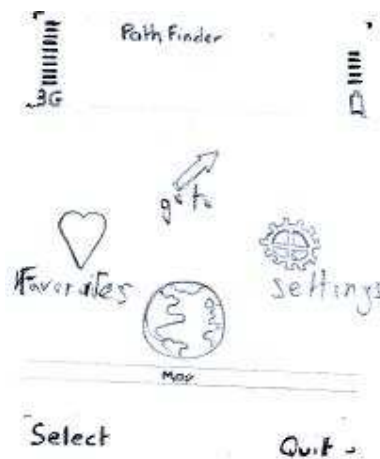
With all those ideas in mind, we drew the first prototype of our interface, a Lo-Fi prototype, drawn on paper. We drew a mobile phone N95 first, and put a hole inside, so we could simulate the screens with other papers.



#### 2.4.3.1) Main menu

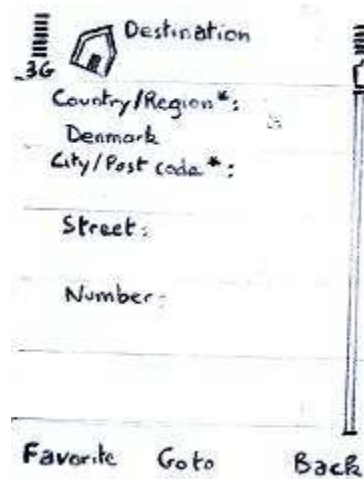
Important part of the interface, the main menu is the first screen of the application, proposing several menus. To keep the application simple, we only propose four menus, because it is sufficient for the application to be efficient.

Since this menu is quite important, and since it is the milestone of our interface, we wanted to make it different than the other menus. That is why; we thought that a menu with four icons that rotates would be nice and user-friendly. Of course we could not represent the rotation on the paper, so we just drew the four icons.



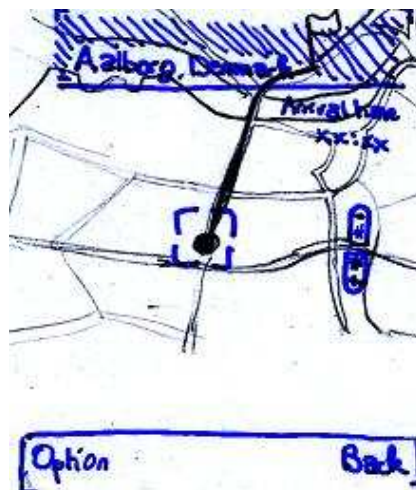
### 2.4.3.2) GoTo menu

When the user selects the GoTo icon, the application brings him to the destination form. This screen just asks the user for the address. First, we chose an arrow to symbolize a trajectory. The user just selects each field with the arrows of the mobile phone, and then type the information needed.



If the user simply wants to go to this destination, then he can press the middle button (for Goto). If he thinks that this address should be in his favorites, then he should press the left button, which would lead him to the favorite menu. At any point, the user can also go back to the previous screen.

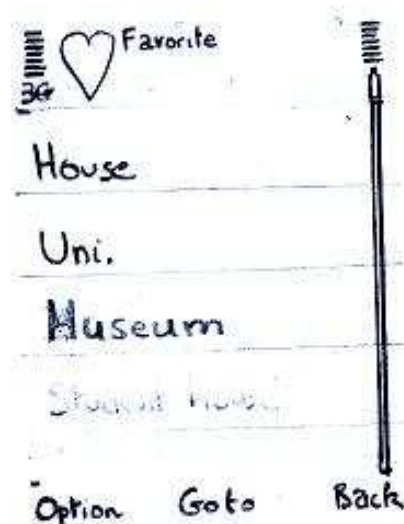
If the user selects "GoTo", then the application calculates the best path and displays it on a map view.



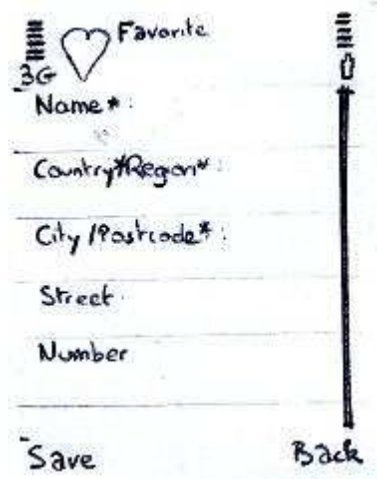
This screen offers several options, like zooming in or zooming out. The user can also move to the left, to the right, up and down. Some information is displayed over the map view, like the destination address, the arrival time, and the path.

### 2.4.3.3) Favorite menu

This menu appears to the user when he selects the favorite icon in the main menu. We chose a heart since it usually symbolized the favorite menu in many known applications. Thus, the user won't be lost, and it won't require any adaptation period. The first screen is the list of favorites already created, with some available options to manage them. In those options, the user can create a new favorite, delete or edit an existing one.



If the user selects the action "Goto", then it will calculate the path to go to the selected favorite. When deleting a favorite, a pop-up shows up to ask a confirmation from the user. When editing or creating a favorite, another screen is proposed to the user, with fields to fill up.



### 2.4.3.4) Settings Menu

Since the program does not have so many parameters to set, one screen seemed enough for the settings menu. We also chose a usual icon for the same reason we explained previously. We assumed that the best way was to show a list of the parameters that could be changed, as well as

their current values. When the user selects a parameter, the application shows all the possible values that this specific parameter can take. The user chooses one of them and validates. It is a simple way to manage the settings. To exit this menu and save the changes, the user must press the left button to save.



#### 2.4.3.5) Map Menu

The map menu is quite easy to understand. It is simply a map, like google maps. We used the globe icon to symbolize it since this is the first view you have on Google Earth, or Nokia Map. The user can zoom in, zoom out, move the map to the left, to the right, up, down, etc.



#### 2.4.3.6) Some remarks on the menus

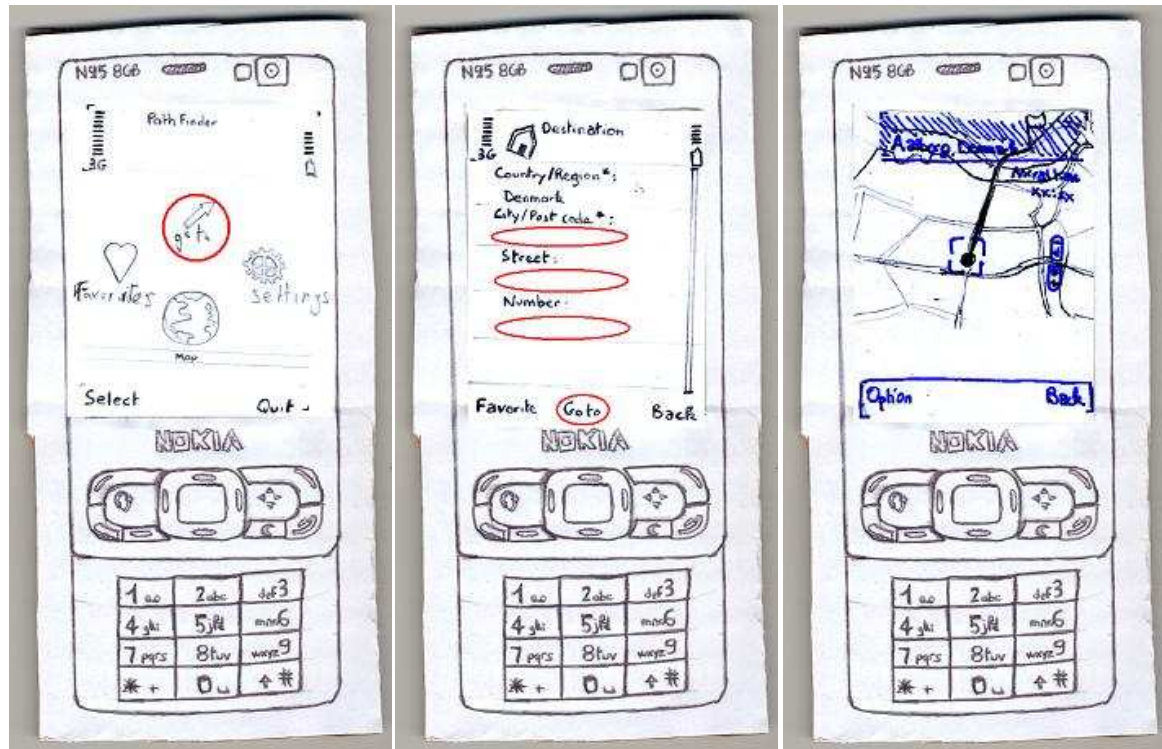
As you can see, the menus and icons are based on the Nokia icons and Nokia menu style. This makes the programming of the interface easier, but also ensures that the application will not seem too different from the other application of the mobile phone.

### 2.4.4) LoFi Prototype test and results

#### 2.4.4.1) The test tasks

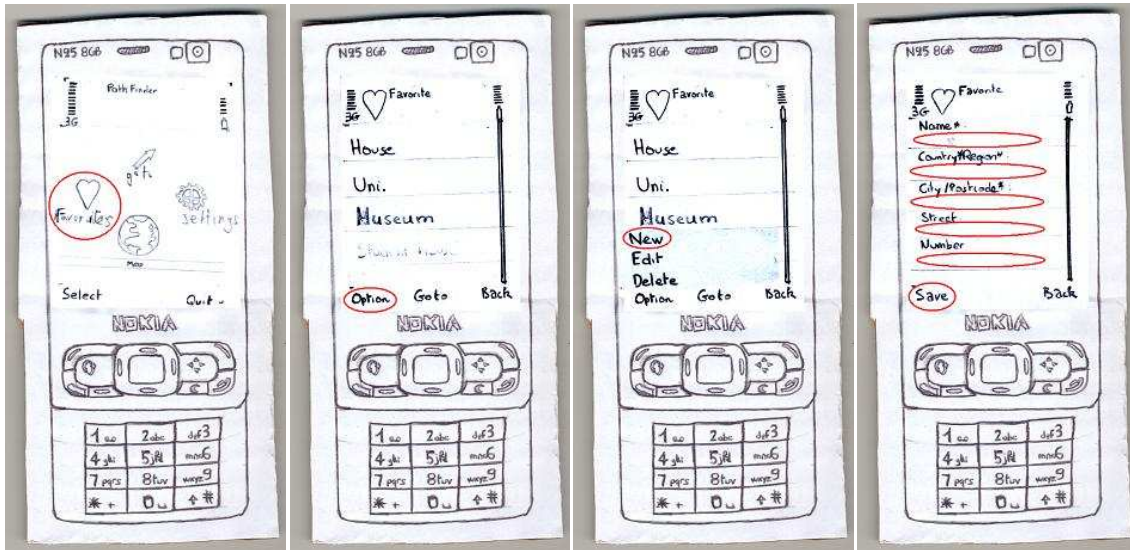
The test proposed only four simple tasks. The purpose was to see if the most common tasks were intuitive for the user.

The first task was to go to a specific destination, by typing its address. Here is the chain of screens the user had to go through.

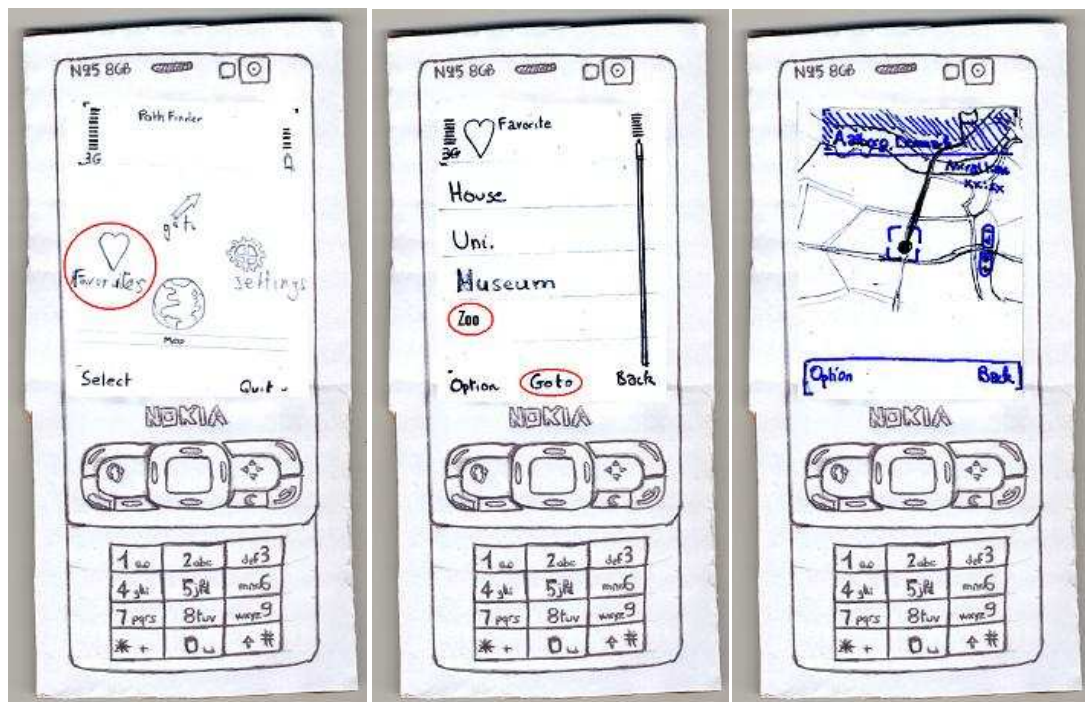


The second task was to create a favorite for a specific address.

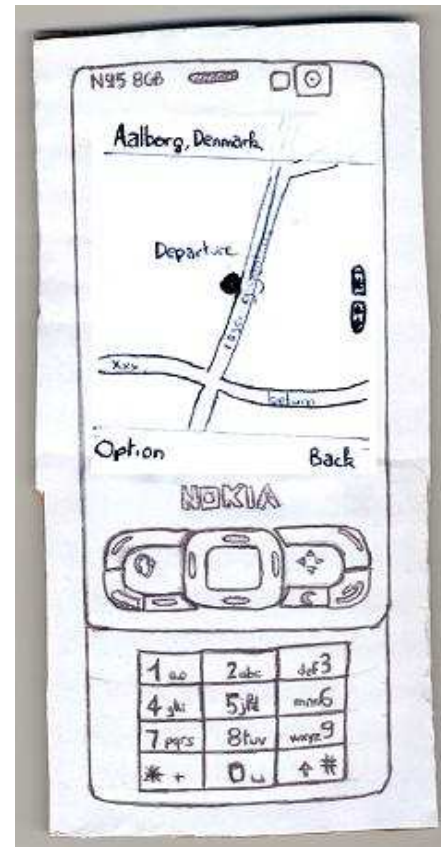
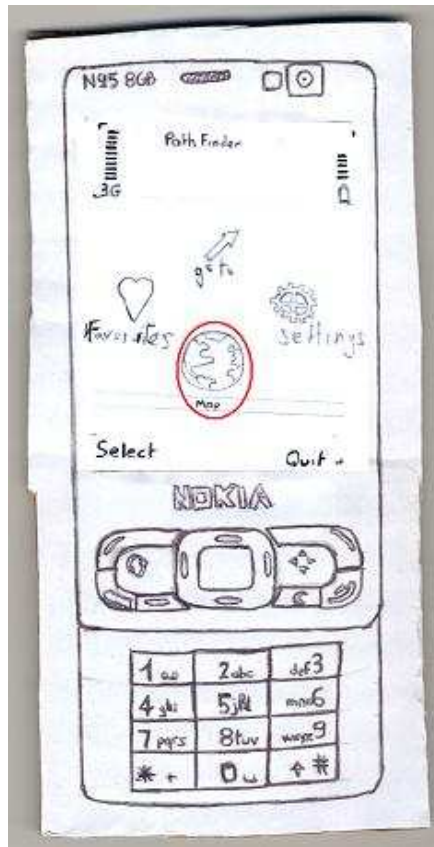




The third task was to use the new favorite to find the path without typing the address again.



The fourth and last task was to have a map view of Aalborg city center. It involved playing with the map menu and with zoom in and zoom out option. Indeed, you first need to zoom out to better identify your position on the city map. Then, zoom in again on random areas.



#### 2.4.4.2) Results of the test

First of all, we performed test in a university room especially made for project testing. Indeed, we took advantage of a place equip with video cameras, microphones, and so on. Plus, the user who tests the project is set in a comfortable couch so that he feels in the best situation to test a design without any pressure.

In general users did not have any difficulties to find out the way to complete the tasks. However, several test-persons did the same mistakes, which pointed out that some elements were not as clear as we thought they were. In order to learn from those mistakes, and to improve our interface, we listed the main problems encountered by the test-persons. We ended up with two main issues.

- Icons in the main menu were a little confusing. The icon for the map and GoTo seemed to confuse the test-persons. Plus, the map icon is already used by Nokia to symbolize the web browser.
- In the GoTo menu, there is an option called "Path mode", which seem to confuse the users. Indeed, they seem not to understand the idea behind this name.

Thanks to those tests, we can improve the interface, and test it again with the HiFi prototype.



### 2.4.5) Questionnaire on the LoFi prototype

Once the users have passed the tests on the LoFi prototype, it is relevant to study how users appreciate it. Thus, we have performed a questionnaire with the few users, but it still allows us to point out the application design inconvenience. [Appendix 1]

We asked the 4 people (3 females and 1 male) aged between 20 and 25 years old to answer our questionnaire about the four tasks we enounced previously. As a reminder, here are the results we recorded:

Task 1: 3 mistakes

Task 2: 3 mistakes

Task 3: Without mistakes

Task 4: Without mistakes

To answer our questions, the users had the help of a scale going from 0 to 5 (from *not at all* to *very much*):

The application is easy to use:

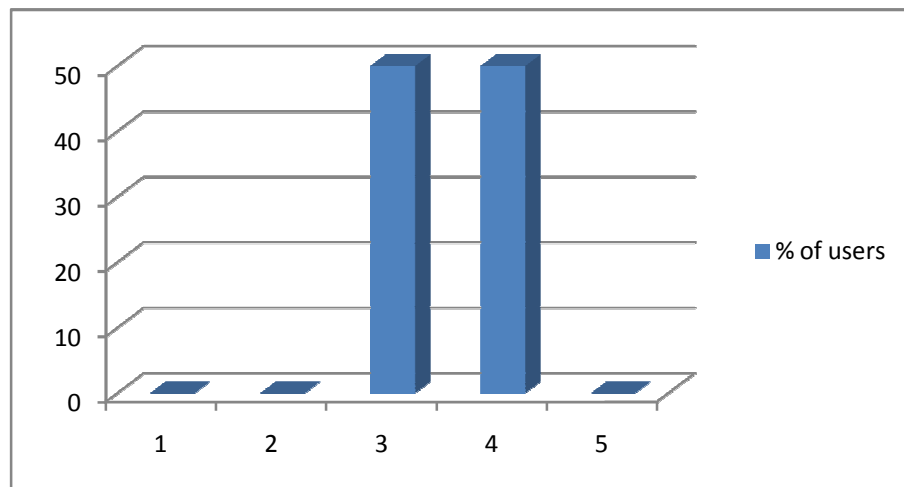


Figure 23 - Answer to the question: "Application is easy to use"

Design is user-friendly:

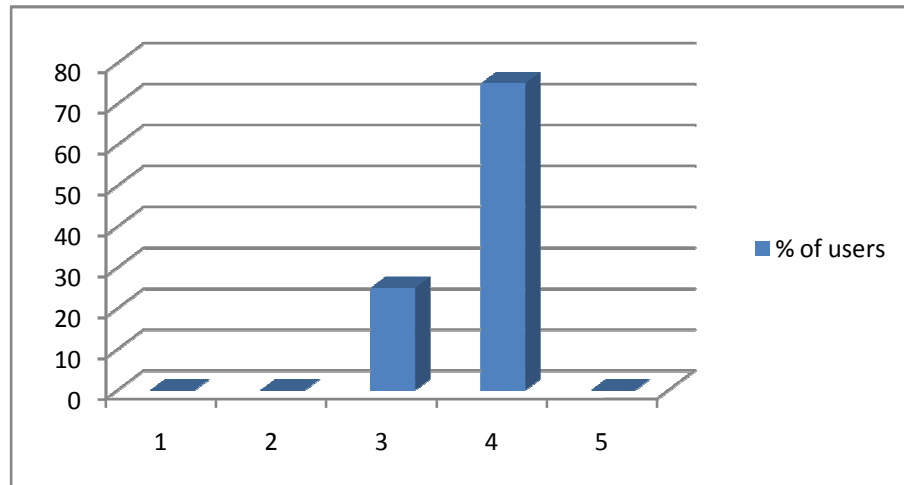


Figure 24 - Answer to the question: "Design user-friendly"

All features are pertinent and useful:

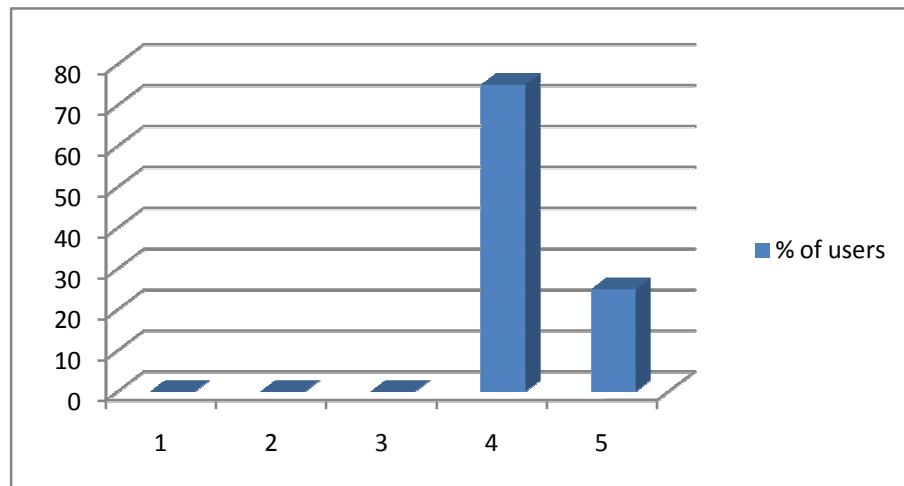


Figure 25 - Answer to the question: "All features are pertinent and useful"

Interfaces are logically organized:

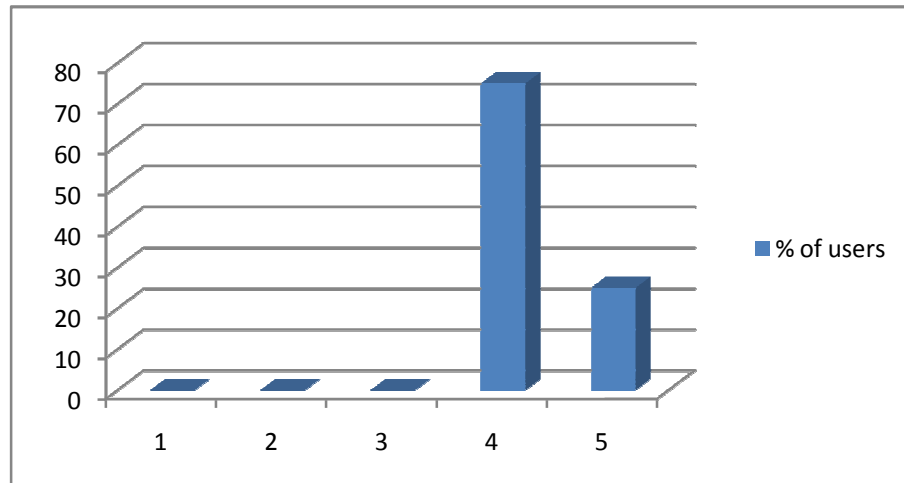


Figure 26 - Answer to the question: "Interfaces logically organized"

You would have liked to have more information:

Yes: [ 25% ]      No [ 75% ]

You would use such an application on your mobile:

Yes: [ 75% ]      No: [ 25% ]

And here are the comments we gathered after the little interview we did:

- Put names in the Main Menu (the user didn't realize that the main menu is a rotation menu).
- Main menu confusing.
- The function of Path Mode isn't clear.

Since all the answers we had are all in the range going from 3 to 5, we concluded that users are satisfied but few things remain to be improved as we have been told in the comments. Thus, the Hi-Fi prototype should correct the mistake the users pointed out while the Lo-Fi prototype tests.

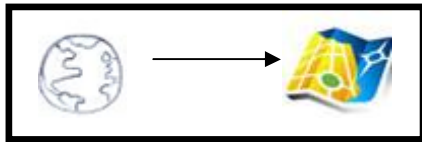
#### 2.4.6) HiFi Prototype

As part of the designing cycle, after testing the LoFi prototype and collecting all the results, it was time to do the HiFi prototype, which is a prototype closer to the final version, and more evolved. As

we said before, we had 2 main problems in the LoFi prototype. First of all, we managed to solve these problems.

#### 2.4.6.1) Solving the LoFi issues

After some discussion, we decided to put off the globe icon for the map menu, and we put a paper map icon instead. The actual icons of Nokia mobiles have this type of icon for the map. You can see the evolution of this icon in our prototype in the following image.



We thought also that changing the icon will not be enough to solve this issue. That is why we decided to change the name of “Map” to “View Map”, which seemed a more explicit label. The word “View” describes the function more precisely, which makes it easier for the user to make a difference between the functions “Destination” and “Map”.

Another issue concerned the “Path mode”. The path mode is a mode of display, which is in fact a list of all the instructions that the user has to follow in order to get to his destination. As said before, the label for this function did not seem clear to the test-persons, so we decided to change it. “Text Mode” seemed the better option, because it described the functionality in a better way.

After solving these problems, another issue was the language we would be using for the HiFi prototype. We will discuss that in the next part.

#### 2.4.6.2) Application used on the development of the HiFi

There were several possibilities:

- Visual Basic
- PowerPoint
- Flash
- C#
- Java

The purpose of this prototype was to focus on the user interface, not on the background processes. That is why Powerpoint and Flash seemed the better choices. We chose Flash simply because some of our group members already worked with Flash and had experience in ActionScript.

In the beginning we started to use Action Script 3, without any specifications. We started developing the skin, all the buttons in the mobile, but after some days, we decided not to continue in this version of Flash, simply because it was too hard to implement some functions, and also because we could not run it on the mobile phone.

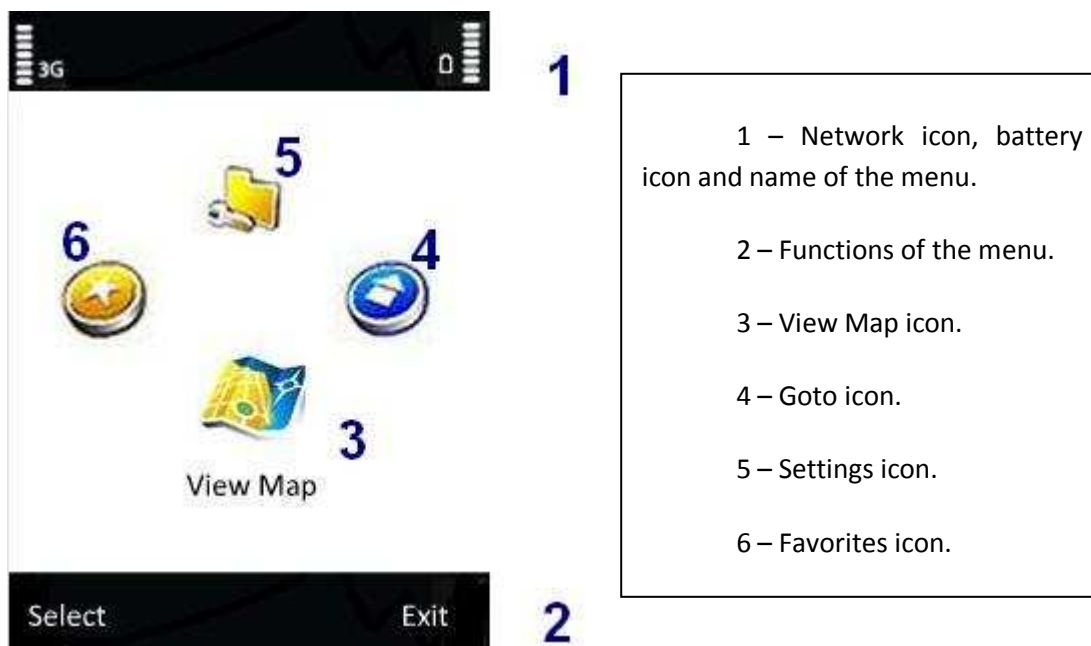
We started to explore and learn how to work with Flash Lite, a version of Flash, specially made for mobile phone application. Moreover, with this version of Flash, we did not have to implement all the buttons in the mobile, as they were already implemented. So in the end, we had the best

solution in our opinion, because we could concentrate more in the design and interface. In the next chapter, we will explain this with more details.

#### 2.4.6.3) Developing the HiFi Prototype

With all the technical issues solved, it was time to design the interface of what will become our HiFi prototype. As before, we wanted a simple interface, and we followed the screens of the LoFi prototype, with the evolutions explained in the previous parts of this section of this report.

First of all, we put two rectangles, one on the top of the screen and another at the bottom, in all the screens. The reason of this was that we wanted to follow the Nokia interface, with the functions explained bellow, and the name of the application, battery icon and network icon above.



After this step, we started to design all the different screens.

#### 2.4.6.4) Main Menu

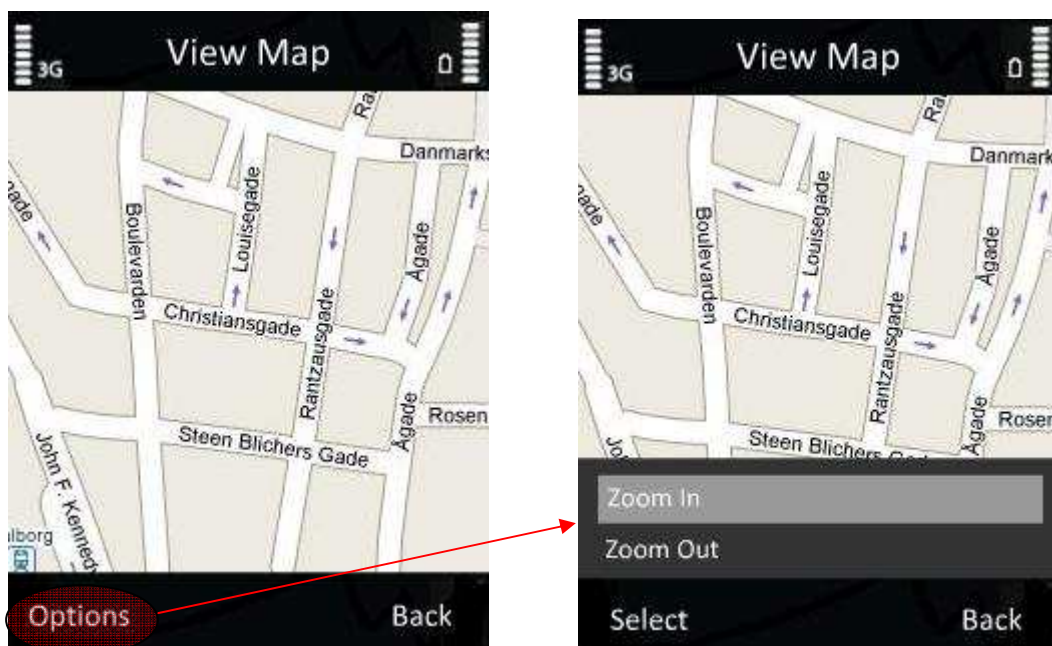
For the Main Menu, we started to develop a menu with a rotation motion, but since this was not a relevant thing for the HiFi prototype, we decided to abandon this development. We used for the main menu some familiar icons and based on S60 Nokia icons.

- **View Map:** we put a simple paper map, since this symbol is more adequate for this kind of function. Moreover, it is the symbol chosen by Nokia Maps software.
- **Goto:** like in the LoFi prototype, we put an arrow to represent this function, since many GPS systems use this kind of symbol for this function.
- **Settings:** many systems, mobile or computer systems, use this kind of icon to represent a function associated with settings or personal configurations. We decided also to follow this pattern in the user interface design.

- **Favorites:** as before, many systems (a good example is internet browsers) use a star to represent favorites. Others use a heart to represent this kind of function. In our case we chose a star, simply because it seemed the most familiar to us.

#### 2.4.6.5) View Map Menu

We wanted to make this function as simple as possible, like the LoFi prototype, but in this case we implemented also the Zoom In and Zoom Out functions. When the user enters in this menu, he will see the previous screen replaced by a map, supposed to be a map centered in the area where he is. The rest of the menu is very simple, with the “zoom in” and “zoom out” inside the options button or accessible by the keys 1 and 3.



#### 2.4.6.6) Goto Menu

When the user enters this menu, the first thing required is to input his destination, field by field. If the user already has his destination in the favorites, he can simply press the Favorite button and go to the favorites to select one of them to be his destination. Otherwise, he has to fill all the text fields, with the name of the country, city, street and number. Since this is a prototype, we just simulated one route, from Denmarksgade to the University, whatever text the user inputs in the text fields. After filling all the fields, he simply has to press the key 5 (as shown in the following screen). The choice of using this key was dictated by the constraints of Flash: the center button had to be used to edit the text fields.



3G GoTo

Country/Region

City/Post Code

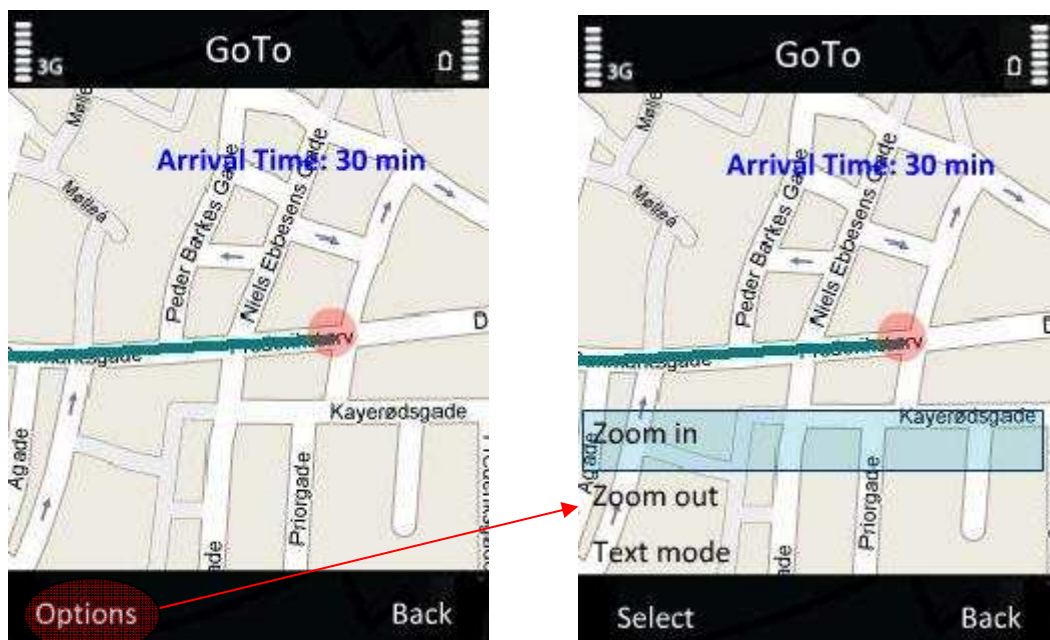
Street

Number

(key 5) Go

Favorite Back

After this step, the user has his route, with the estimated arrival time to his destination shown in the top of the map. To identify where he is we used a red circle, and to identify the route a green line. With this, the user can know which direction he has to take. To give to the user a fuller control of the goto menu, we implemented also the text mode view. For this he has to simply press the Options button and after select the Text Mode button.



Within the text mode view, the user has the path divided into steps (or instructions). Those instructions are displayed as text on the screen, in case he doesn't want to see the graphical representation of the path. We chose to have a simple representation, just with some information,

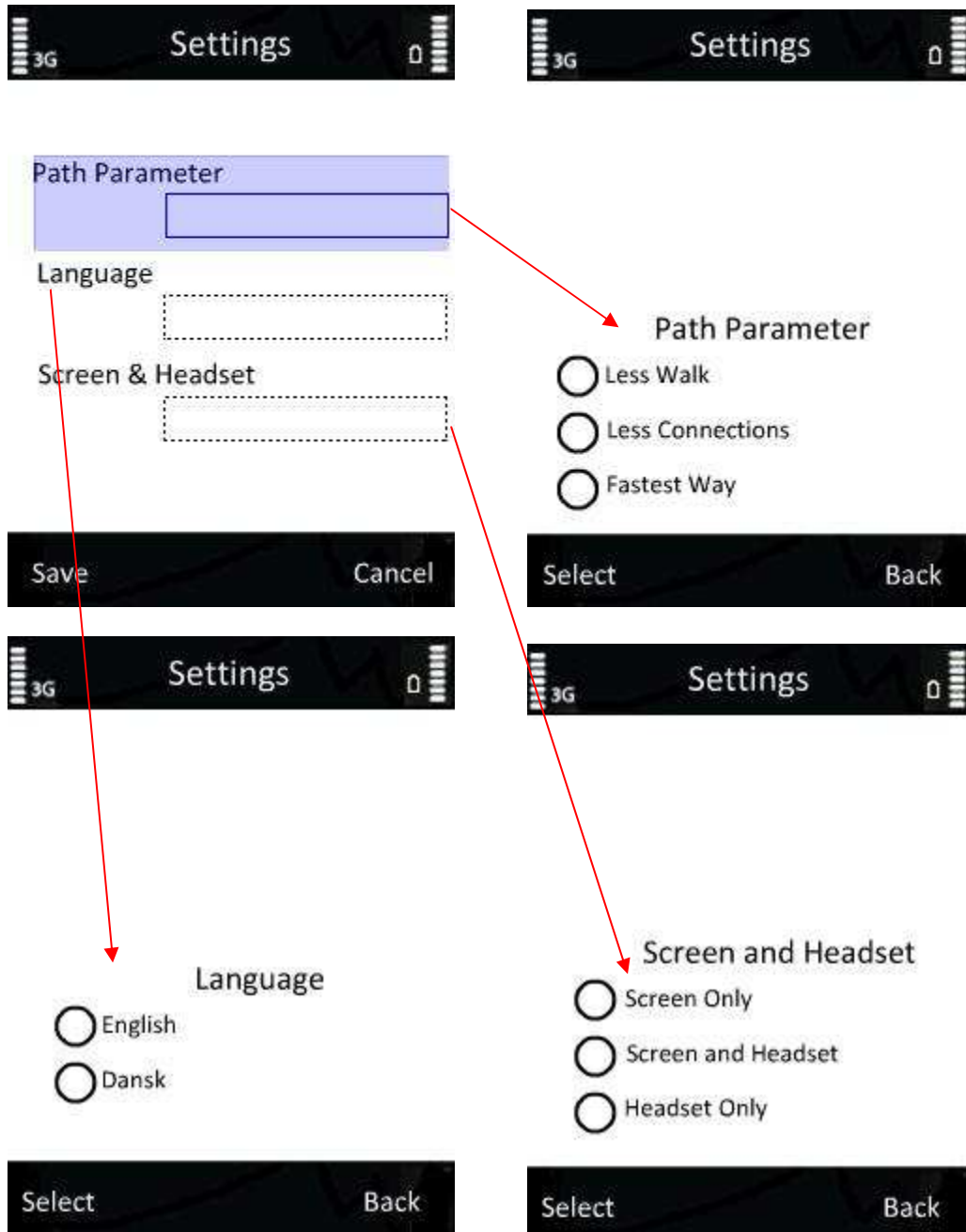
like the street that he has to follow, the direction and an arrow to represent a turn or a direction he has to take.



#### 2.4.6.7) Settings Menu

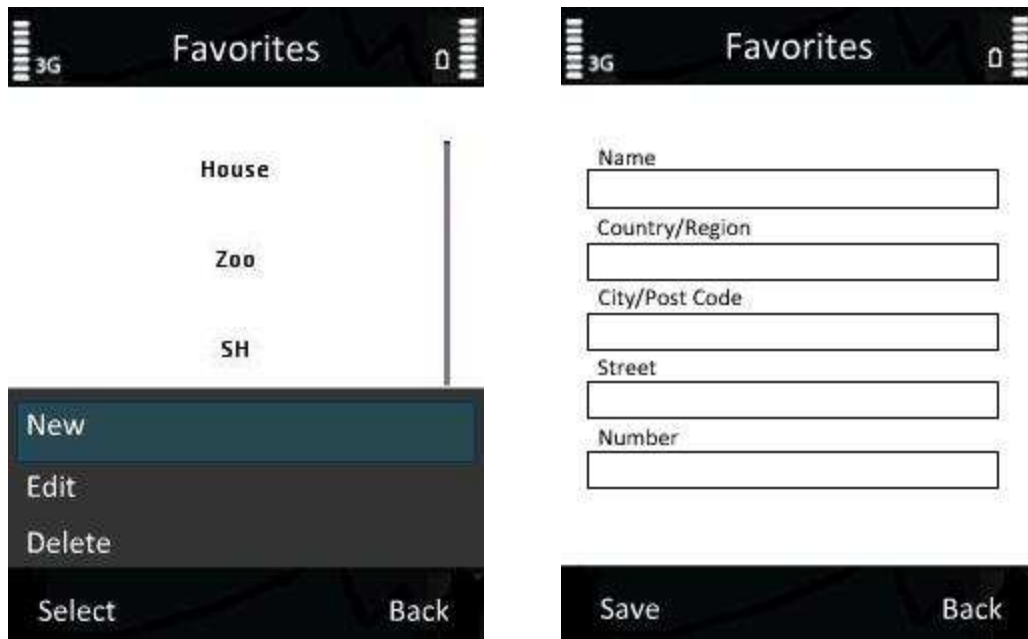
The idea of this menu, as told before, is to give the user some control over the application, like the selection of the language, the preferences as to the kind of way to his destination or how to display the information. This is also an important thing in user interface design, to give to the user some configurations that he can select. In this menu, depending of which function the user selects, he goes to the specific setting that he wants to configure.





#### 2.4.6.8) Favorite Menu

As told before, to give to the user the option of saving some of his destinations, we created a favorite menu, which enables the user to create a new favorite, edit or delete. Besides that, it is also possible to see the route to a favorite.



When the user wants to create a new favorite, he has to fill up all of the text fields (like he would do in the goto menu) including the first one that is the “Name” field (the favorite label). This field will be used to identify the favorite in the favorites list.

After the hifi prototype development, we tested the prototype with the users, with four simple tasks, as we will talk below.

#### ***2.4.7) Testing the HiFi prototype***

In these tests, we presented the test-persons with four small tasks, quite similar to the LoFi tasks:

- Use the application to find your way to University, using the address “Fredriks Bajers Vej 10”.
- Add the address of the University to the favorites. You can give it any name you want.
- Find your way to University, without typing its address again.
- Display a map view of Aalborg city centre on the screen of the mobile phone.

##### **2.4.7.1) First Task**

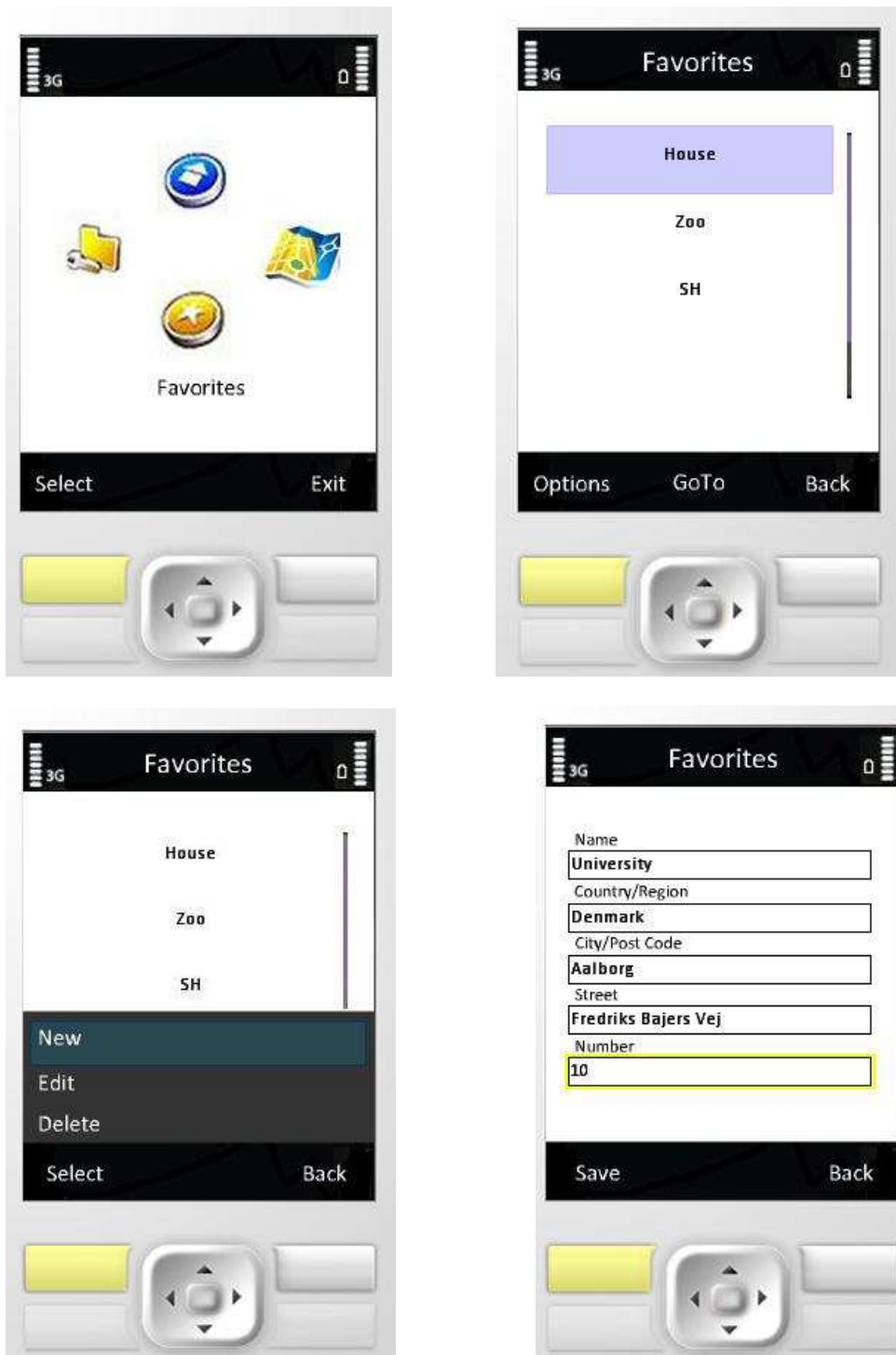
With this task, we wanted to test the main function of our system: see the route to an address that the user inputs, without using favorites.



In this task, each user made between 2 and 3 errors. Most of them were made while filling the text fields, but mainly, this problem is associated with the implementation of Flash Lite. Indeed, because of this version of flash, the user has to press the middle button on a text field before writing, and it was quite confusing for the users. This kind of problems would not be in the real application, which will behave itself properly. The only problem with our interface was between the “goto” and “view map” functions. One of the users went in the view map menu, instead of goto menu.

#### 2.4.7.2) Second Task

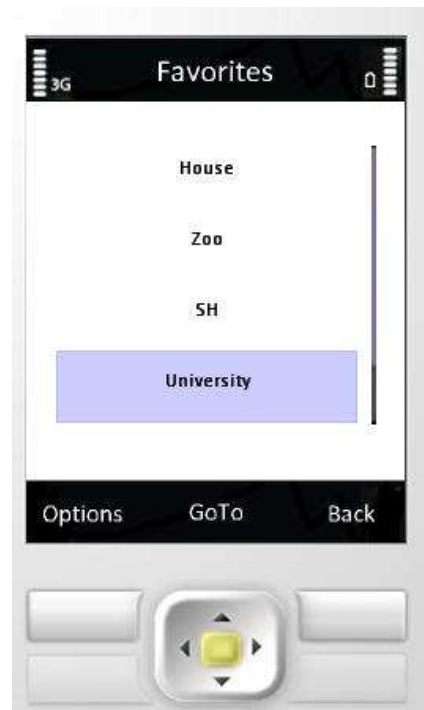
With this task, we intended to test how easy it is to put a given address in the favorites list.



In this test, we had only one user with 2 mistakes. These mistakes were due to the fact that the user did not realize that the favorite list was displayed in the favorite menu, and didn't see the Options button (2<sup>nd</sup> screen).

#### 2.4.7.3) Third Task

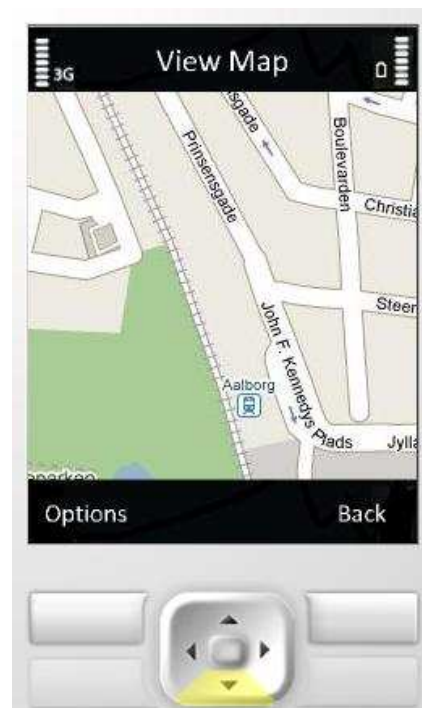
With this task, we wanted to test how easy it is to go to a destination previously inserted in the favorites list. In fact, for this task there were 2 options: go to the "goto" menu and after press the Favorite button, or simply go to "favorites" menu and choose the destination.



In this test, one user made 1 mistake only. He went in the “view map” menu instead of the “goto” or “favorites” menu. Maybe this user was still confused about the difference between the “view map” and “goto” functions. But, after he realized his mistake, he did the rest of the task pretty well.

#### 2.4.7.4) Forth Task

In this final task, we intended just to test how the user will manipulate the map, without inserting any destination or favorite.



With this task, the users did not make any mistakes. One of the users also tested the “zoom in” and “zoom out” functions.

After having the test, the users had to fill in a questionnaire. We are now going to have a look to what we could extract from that.

#### 2.4.8) Questionnaire on the HiFi prototype

- **Population:** 3 users (3 Female)
- **Age:** between 21 and 30
- **Different kind of users** (2 Bus users / 1 car driver)

The HiFi prototype has now been tested on several users. It is now relevant interview them in order to have their feedback. Therefore, we asked the 3 people (3 females) aged between 21 and 30 years old to answer a few question after accomplishing the four tasks we enounced previously. [Appendix 1] As a reminder, here are the results we recorded:

Task 1: 8 mistakes

Task 2: 2 mistakes

Task 3: 1 mistake

Task 4: Without mistakes

The users had to answer the questions with the help of a scale going from 0 to 5 (from *not at all* to *very much*):

The application is easy to use:

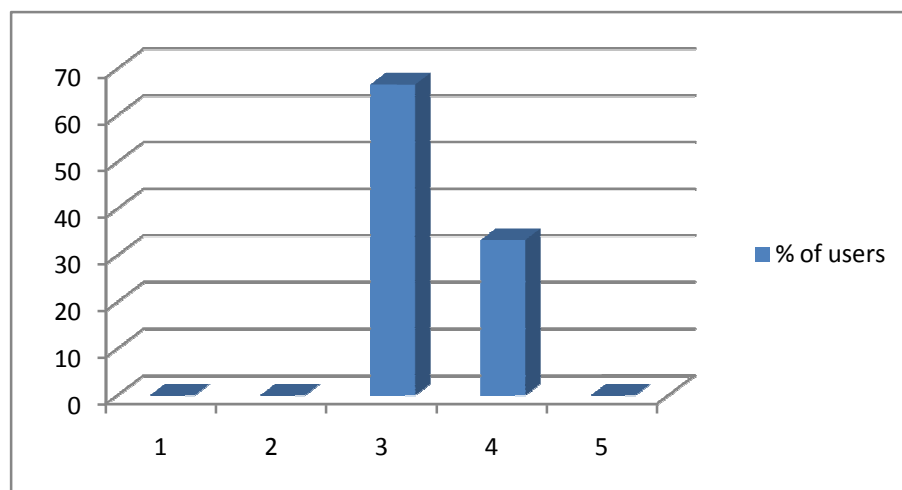


Figure 27 - Answer to the question: "Application is easy to use"

Design user-friendly:

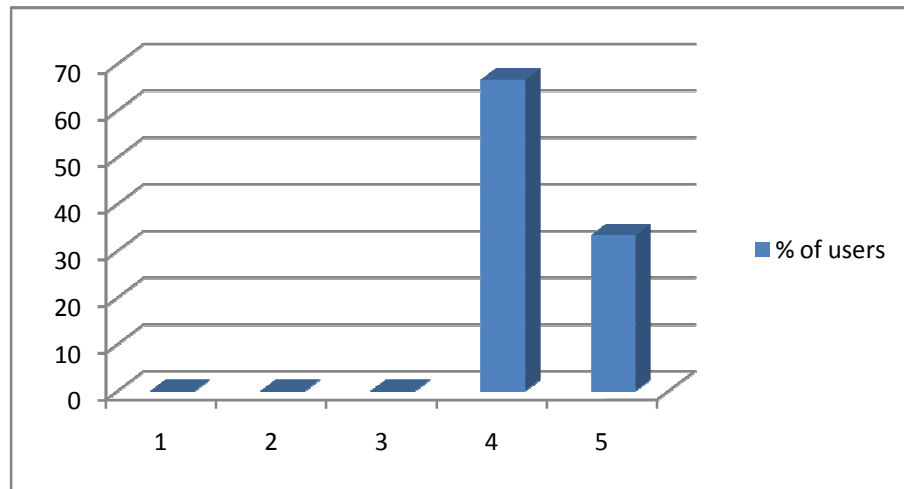


Figure 28 - Answer to the question: "Design user-friendly"

All features are pertinent and useful:

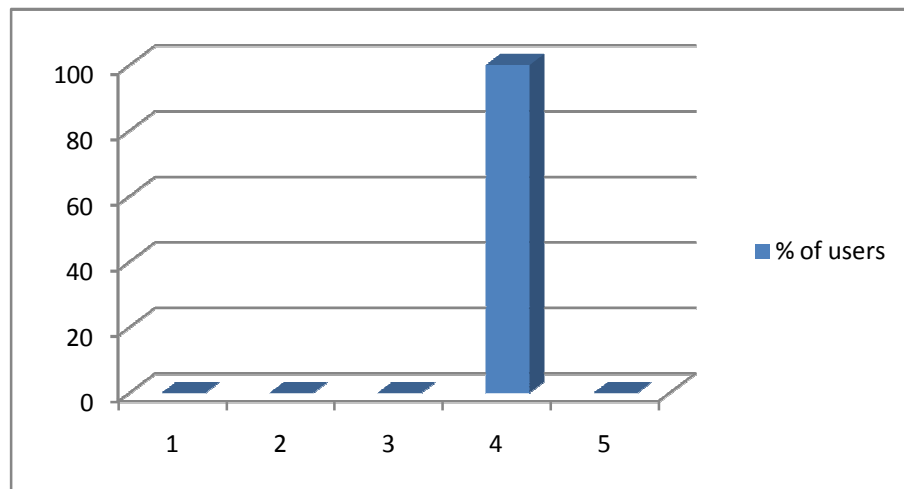


Figure 29 - Answer to the question: "All features are pertinent and useful"

Interfaces logically organized:

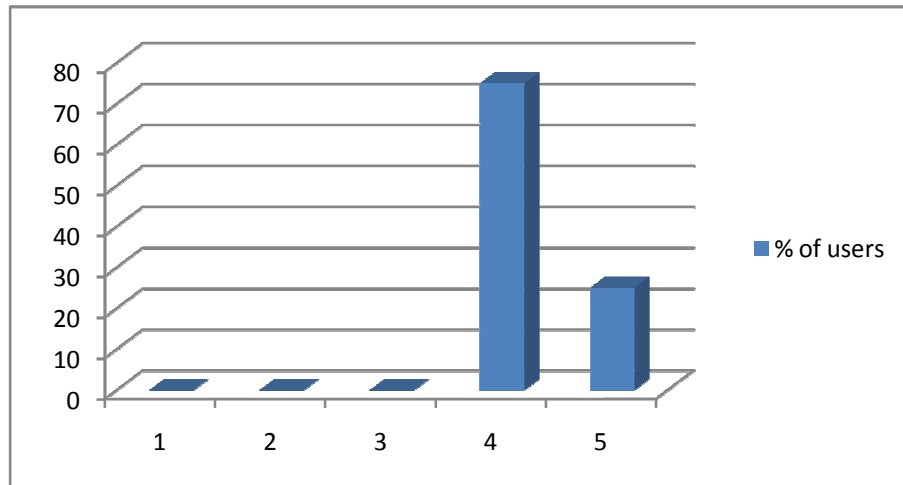


Figure 30 - Answer to the question: "Interfaces logically organized"

You would have liked to have more information:

Yes: [ 0% ]      No [ 100% ]

You would use such an application on your mobile:

Yes: [ 66.6% ]      No: [ 33.3% ]

And here are the comments we gathered after the little interview we did:

- Some problems writing in the text fields (Flash Lite problem)
- Favourite list was not so clear
- The key 5 to validate the destination was not so clear

Thus, this prototype had some surprising results for us, both positive and negative. First of all, we didn't expect users to find the prototype so hard to use. In fact, we encountered some problems during the tests concerning Flash Lite API. Plus, users faced issues with the Favourite menu since they didn't realize it was a list of favourites. In fact, this problem has an origin: when we did the prototype we saved 2 favourites address in the program, just to simulate a program which would have been used before. In the end, we concluded that this idea wasn't so good, since it's the first time they all were using the system and expected an empty favourite list. Nevertheless, this it will not happen in the final program.

The rest of the results were quite positive. Thus, we concluded that finally we did a very functional GUI with clear indications.



### ***2.4.9) Conclusions of the tests***

In general, we had a satisfactory opinion about the user tests. We found only two main issues with our design:

- Some users confused “view map” function with the “goto” function. This was an error that came from the LoFi prototype. We made a correction, but the error still exists. However, this time just one user made that mistake, so we concluded that it was just a matter of the user getting used to this system.
- Another problem was in the favorites menu. One of the users did not realize at first that there was a list of favorite in the favorites menu. We thought that is not a big problem, since in the real implementation, the favorites list will be empty in the beginning, and the user will add them himself.

### ***2.4.10) Design of the final interface***

Now, we will describe the real interface we designed for the mobile phone application. After the testing of the HiFi prototype, we decided to draw the final design for our user interface. We gathered the result we received from the Lofi and Hifi prototypes, and analyzed them. We drew conclusions from the mistakes the user made during those tests, and we implemented the user interface in the mobile phone application.

As for the LoFi and Hifi prototypes, we will describe the different screens we made, and we will explain the improvements we made compared to the Hifi prototype. But first, we will explain how the user can move from one screen to another.

#### **2.4.10.1) Application screens tree**

In order to see clearly how the interface is built, we have drawn a diagram explaining the relationships between the screens. On this diagram, we can see clearly how the user can get to a specific screen, and the different paths that can lead him to it. The application is not very complex from that point of view, but there are often several ways to go from one screen to another.

On this diagram, the screens are displayed like a tree. There are labels on the links between the screens: those labels are the names of the commands the user has to select to go from the first screen to the second one. For example, there is a link between the “Favorites List” screen and the “Favorite Deletion” screen. On that screen, it is written the keyword “Delete”, which means the user as to select the “Delete” option to go from the “Favorites List” screen to the “Favorite Deletion” screen.

At the top of the diagram, and as the root of the tree, there is the main menu, giving the user access to the four main functions of the application. It is the starting point of the application. We can see that the “View Map” screen only displays the map, but doesn’t let the user go to another screen (of course the user can go back). The settings menu only leads to a pop-up screen “Settings editing”, which permit the user to change the value of a parameter. The “Favorites List” screen leads the user to the “Favorite Deletion”, “Favorite Creation” and “Favorite Editing”. Finally, it is important to note that the user can go from the “Destination” screen to some of the favorites screen. For example, the

user can go from “Destination” screen to “Favorite creation” screen, if he wants to add the address he has just typed into his favorites list. We will explain the available options for each screen in the following parts.



Figure 31 : Tree of the mobile phone application screens and their relations to each other.

#### 2.4.10.2) Main menu

We kept our initial idea of making a rotation animation for the main menu. The only thing we changed was some of the icons, because the icons we used in the tests were a little bit confusing for some of the users. The main menu is maybe the most original part of the user interface, which means it can also be the most confusing. That is why we tried to make it as intuitive as possible, so that the user can clearly understand how to interact with this object.



Figure 32 : Main menu of the mobile phone application

The user can use the arrows of the keyboard to make the four icons rotate. He can also use the numbers 4 and 6 on the keyboard to get the same effect. We will speak more about this menu in the implementation part, since we had some problems to develop it.



Figure 33 : Main menu after rotation to the right. The user can press OK to view the map.

As it is easily seen, the icon in the front is bigger than the other, to enhance the fact that it is the selected one. Also, the label associated to this icon is displayed just below it. The other icons don't have their labels displayed. We thought that the icons were sufficiently explicit, and that the user would memorize their meaning quite quickly. Anyway, by rotating the menu, the user can have a look at the other labels.

#### 2.4.10.3) Settings menu

When the user selects the settings icon, it will lead him to the settings menu. This menu is quite similar to the HiFi prototype, since we did not make many changes.



Figure 34 : The settings menu.

The settings menu displays the list of all the parameters that can be changed, as well as their current values. As for the HiFi prototypes, the user can select one of those parameters and change its value.



Figure 35 : The user is editing the parameter "sound".

When the user selects a parameter and presses the middle button, it opens a pop-up. This pop-up contains all the values that the parameter can take. The user selects the values with the keyboard arrows and presses the middle or the left button to validate. The mobile application will then display the settings menu screen again. The parameters changes will only be saved when the user presses the "save" button. If the user presses "back", the parameters changes will not be taken into account.

This menu is quite intuitive, and similar to many other applications. That is why we did not make many changes compared to the prototypes. There is no pop-up asking for confirmation when the user saves the parameters, because it seemed too annoying and useless. Without any pop-up, the application is more efficient, and the user can change its parameters quickly.

#### 2.4.10.4) Favorites menu

This menu proved to be well-designed thanks to the test. We did not change anything compared to the hifi prototype. However we did have to design the screen for the favorites list when there is no favorite. We chose a very simple screen, with the message "no data" in the middle of it. The user is then well aware that there is no favorite yet. The only button available for the left key is the "add" function. Indeed, the user cannot edit or delete any favorites.



Figure 36 : There are no favorites created yet.

When the user selects the “add” function, the “add favorite” screen is displayed. It is quite the same as the hifi prototype screen. The user has to fill in a form with the address and the name of the new favorite. We decided to give some default values to the fields “Country” and “City”. Indeed, the application is supposed to be limited to the region North Jutland in Denmark, and our prototype is supposed to work only for Aalborg (for reasons explained in the conclusion of this report). That is why the default values of “Country” and “City” fields are “Denmark” and “Aalborg”.





The figure consists of two side-by-side screenshots of a mobile application interface titled "New Favourite".

The left screenshot shows the form with the following fields and values:

- Name: (empty)
- Street: (empty)
- City: Aalborg
- Country: Denmark

The right screenshot shows the form with the following fields and values:

- Name: Home
- Street: Blegkilde Alle 8
- City: Aalborg
- Country: Denmark

Both screenshots show a "Save" button at the bottom left and a "Back" button at the bottom right. The status bar at the top of each screen shows the time (8:42 am and 8:44 am respectively) and a signal strength indicator.

Figure 37 : Creation of a new favorite.

As in the prototypes, the user fills in the form and presses the “save” button. Once again, there is no confirmation to be given, since the user can always edit or delete the favorite if he made some mistakes in one of the fields in the form.

Once the favorite is created, the application displays the favorites list.



Figure 38 : Favorites List and the available options.

When the user selects the “edit” option, it enables him to change the name or the address of the selected favorite.



Figure 39 : Edit the favorite // Pop-up the user gets just after selecting the "delete" option.



If the option “delete” is selected, then the user interface asks for a confirmation, because this is a definitive action (there is no way to get the favorite back in memory after this). This confirmation is asked via a pop-up “are you sure”.

#### 2.4.10.5) View Map

When the user selects the “view map” option, the application displays a simple google map of the surroundings. The user can use the arrows of the keyboard to move to the North, South, West or East. He can also use the keys 2, 4, 6, and 8 for the same effects. The keys 1 and 3 enable him to zoom in and zoom out. This is quite a normal interface for a map display, and quite intuitive.



**Figure 40 : Map of the surroundings, with the position of the user marked on it.**

We also decided to add a marker on the user's position. During the implementation, we also see that the GPS signal was lost sometimes. Thus, we thought it could be very useful to display an icon telling the user if the GPS signal was still received by the mobile phone. The icon is shown in the top right corner of the map image. A green icon means the mobile phone is synchronized with the GPS, and a red icon means the signal is lost. This is a very useful feature that we did not conceive in the Hifi prototype.

#### 2.4.10.6) Destination menu

When the user selects the destination icon, the program asks first for a starting and a destination address. Actually, this screen is quite different from the Hifi prototype screen. Indeed, we decided that the application should be able to run even if the user does not have a GPS on the mobile phone. In this case, the user must give the system a starting address. That is why we added a checkbox on the top of the screen.

This checkbox determines if the user wants to specify a starting address or not. If no GPS is detected on the phone, the system will ask for a starting address. Otherwise it depends on the state of the checkbox.



Figure 41 : Start from current position or Start from an address.

When the user deselects the checkbox, the program adds three fields to get the starting address. The fields “starting city” and “starting country” are set by default to “Aalborg” and “Denmark”.

This allows the program to be more flexible. It can work on a mobile phone that does not have GPS embedded.

When the user has given his destination address and his starting address, he has to select “options” and then choose between “Go To”, “Add to favorites” and “Go to favorites”. This is quite different from the hifi prototype screen. Indeed the hifi prototype has no “Options” button, but instead, it has “Add to favorites” and “Go To”. The “Go To” button was associated to the middle button. This is not the case in the real application, and this difference is due to several different issues.

First, we faced an unexpected technical issue, for the placement of the commands. We wanted to associate the “Go To” command to the middle button, and the “Add to favorites” button to the left button. It turned out that it is not possible on the S60 series. It is indeed not possible to define a command only for the left button or the middle button. The commands are associated to both buttons. When there are several commands associated to those buttons, the word “Options” appear on the left bottom of the screen, meaning that the user can access those commands via a “options” pop-up. We used that solution for our two commands “Go To” and “Add to favorites”.

Second, we decided that the user may want to just go see his favorites list, without coming back to the main menu. Since we had already several commands associated to the “Options” menu, we thought that it would not be a problem to add a third one, to increase the flexibility of our program. We thus added the command “Go to favorites”, which leads to the favorites list screen.

#### 2.4.10.7) Path display screen: Map mode

This screen displays the google map of the surrounding. The current position of the user is displayed by a google marker, if a GPS has been connected to the mobile phone. As for the “View Map” screen, a GPS icon is displayed on the top right corner of the screen. A green color indicates that the GPS signal is okay, and a red color signifies that there is no signal anymore. Moreover, there is some more indication on the screen, like the arrival time. The path the user has to take is displayed in a two different ways:

- A red line indicates that the user has to walk, following the lines. The line goes only where a pedestrian is supposed to be able to go.
- Black dots indicate that the user has to take the bus. The black dots represent the bus stations the bus will pass by. When a dot is linked to a red line, it means the user has to take or get off the bus.



Figure 42 : Destination Path screen.

As the user is moving, the map will be refreshed, to fit the new position of the user. Thus, the user can see his updated position on the map, which makes it a lot easier to follow the path displayed.

#### 2.4.10.8) Path display screen: Text mode

This screen can be accessed by the command “Text mode” available on the Map mode. The user can switch from one screen to the other using the commands “Text mode” and “Map mode”. The text mode offers a description of the path, step by step. The path is thus divided into several instructions, like “turn left”. For each instruction, there is an icon which enables the user to quickly see what kind of instruction it is. The text next to the icon gives more details: each text is composed by “turn left”, “turn right” or “go ahead”, as well as the name of the street the user has to take.



Figure 43 : Text Mode.

#### 2.4.10.9) Conclusion on the user interface

In this part of the report, we have been presenting the thinking process and the design development concerning the user interface. We have first explained the main ideas and constraints that we had to work with. Then we explained how we designed the LoFi prototype, and the results of the tests we ran with it. Taking into account those results, we described the process of developing the HiFi prototype, and testing it. Finally, we described the interface we realized for the application.

As it has been noted, several changes have been made between the LoFi prototype and the final design. This is due to the tests results we got from the LoFi and HiFi prototypes, but also because there were some elements we could not implement in the final user interface. Some other items were changed because it did not seem relevant anymore, or because we got some better ideas after the HiFi prototype.

However, the tests we ran on the prototypes were very helpful, and gave us some very useful feedbacks that guided us for the final design development. It allowed us to design a user-friendly and

efficient interface. The final tests we ran on the mobile phone application will be described in the Testing part of this report.

These results also helped us to define the graphical requirements of our program, and thus, to choose the programming language.

## 2.5) Programming language choice

Before starting the development phase, we spent some time to select the appropriate developing language. Even if all of us already developed some mobile applications on mobile phone using C# language and Windows Mobile, we were not familiar with Nokia mobile application developing tools and we tested them before making any decision. [xvi - Wikipedia, 2008]

As the N95 implements a Symbian (S60) operating system, almost all of the applications are developed using Java and C++.

For this reason, we installed the tools Carbide (C++ IDE) and Eclipse (Java Me and Java Sdk) and coded different programs using both languages.

### 2.5.1) C++,Java and Python

#### 2.5.1.1) C++

**Advantages:** Object Model, faster, portable on all S60 platform devices, all the functionalities of the mobile phone can be accessed, a lot of code examples can be found on Nokia website

**Drawbacks:** Not portable between different mobile operating systems, less productive language

#### 2.5.1.2) Java

**Advantages:** Object Model, Language already known by all the developers, porting form other platforms, efficient for web application

**Drawbacks:** Less functionalities/API than C++, slow

#### 2.5.1.3) Python

**Advantages:** Object Model, porting form other platforms, efficient for fast graphic applications development, enable to develop a graphic application using the same patterns as the native S60 applications.

**Drawbacks:** Language not known by all the developers, less portable than java

### ***2.5.2) Our choice***

As we had 4 months to develop the application, we preferred to use a language which is more productive than C++ such as Java or Python. As Java was a familiar language for all the developers and was also more portable than Python, we chose to develop the mobile phone application in java. We then checked that programming in Java wouldn't restrict the functionalities of the application. For this cause, we made a list of the application sensitive functions to implement:

- Rotating Menu: Using SVG icons, the slow speed of Java shouldn't be a problem. We succeeded to display an animated menu.
- Displaying a map:  
For reasons developed previously in the analysis, OpenStreetMap doesn't work well with mobile phone applications concerning map pictures retrieval. However Google Map seems to be the best solution for map pictures. Thus, we will now use both OpenStreetMap for the streets coordinates, and Google Map to display the map.
- Favorite settings: the user should be able to save his favorites address, languages, etc.
- Connecting to the internet: Using Java functions, connecting to a website is reliable and fast.
- Development tools: Java Me and Eclipse tools are efficient, reliable and free tools to develop mobile phone java applications are available.

After this analysis of the language options, we conclude that using java will enable us to develop in the allotted period an efficient and reliable application without any speed concern.

## **2.6) Code structure**

### ***2.6.1) Constraints for application development***

First, we should summarize the role of this application. Indeed, the mobile phone is the only access point to the application. Its interface is the only user interface, since the server is not meant to be used directly. It was thus important to focus on the design and take care to follow the HiFi prototypes.

Another important constraint was that the mobile phone, even the Nokia N95 (on which we ran our tests), is not as powerful as a computer. This means that we had to delegate as much calculation as possible to the server, which is able to compute quickly those operations. This also implies that the mobile phone must send information to the server, but not too much, so the transfer time is not too long.

### ***2.6.2) Global structure of the application***

The application is divided into different packages. We will explain briefly the purpose of each package, and the principal relations between them. There is a class diagram in the [Appendix – 4] of this report.



### 2.6.2.1) ApplicationGlobal

The role of this package is to manage the entire application. It contains the Main of the application, and the classes Language and ScreensManager. We will explain more deeply the roles of such classes.

- Main

This class contains the main of the application. It creates all the screens, as well as the records managers, the screen manager and the language database. It then asks the screen manager to show the menu screen.

- ScreensManager

As said previously, the Main class creates all the screens at the beginning of the application, which means that all of them are instantiated. To show a specific screen to the user, the program has to set this screen on the “display” object. We use the following method to do that:

```
Main.display.setCurrent(Main.screenMenu);
```

Where “display” is a display object used to show something on the screen of the mobile phone.

As a matter of fact, it seemed useful and efficient to create a class in order to show the right screen at the right time. The ScreensManager class contains some constants: each constant is linked to a screen. By calling the method showScreen of the ScreensManager, with the appropriate constant, another object can summon the screen.

```
Main.screenManager.showScreen(ScreensManager.SCREEN_FVT_EDIT, true);
```

Another purpose of the class is to get the previous screen. We indeed wanted to let the user have the possibility to go back to the previous screen. That means we needed a way to keep in memory the stack of the screens the user has seen. That is also done in the ScreensManager class. It adds each screen to the stack. When the method “showScreen” is called with the SCREEN\_PREVIOUS constant, it gets the last screen in the stack and shows it on display.

```
Main.screenManager.showScreen(ScreensManager.SCREEN_PREVIOUS, false);
```

Each screen is related to a class in the program. Each time a screen is called, the variables of the object called are reset, thanks to the function “refresh”. This method can be found in nearly all other classes.

- Language

One of the important points of our application is that there are several languages available to the user. That means that the current language must be checked each time a text must be printed on the screen. This is not convenient in the code, and it must be done in a clever way if we want it to be easily modifiable. That is why the language class was created. It contains a lot of constant (one for each label of the application), and a method that makes the relation between a constant and a string,

depending on the language selected by the user. For example, we need the label “Back” in one of our screen. We will not write “back” in the code, but rather call the following function, which will return the equivalent of “back” in the language chosen by the user.

```
Language.getString(Language.BACK);
```

The “getString” method will return “Back” if the language is English or “Tilbage” if the language is Danish, for example. This came very handy when we had to add some new languages to the application, because we only had to modify one class. Everything concerning the labels is in the same place, making eventual modifications much easier.

#### 2.6.2.2) Network

The purpose of this package is to ensure the communication between the mobile phone and the server. It contains all the declarations necessary to establish a connection from the mobile to the server, and to exchange information.

- Connection

The Connection class is the class handling the data transfer between the mobile phone and the server. The connection runs in a different thread, in order to receive and send data to the server at any time. Several functions have been made to send either GPS coordinates or an address. If the server receives an address, it will use its geocoding function to get the GPS coordinates of this address. Most of the time, the mobile phone will send the GPS coordinates of the position of the user, and the address of his destination.

The method “receiveXML” will get the result of the server computing. It also calls the method to parse this xml. We will give more details later about the transfer and the uses of this XML file.

- FromAddressToAddress, FromCoordToAddress, FromTo

Those classes contain the data structure to store a starting address and a destination address (FromAddressToAddress), or starting coordinates of the starting point and destination address (FromCoordToAddress). Those objects are used for the transfer of data between the mobile phone and the server.

- Reader and Writer

Those two classes contain the tools to write and read from the connection socket between the mobile phone and the server.

#### 2.6.2.3) Destination

This package contains all the classes related to the “Go To” menu. There is a class for each screen. It will thus manage the screen where the user enters the address where he wants to go, and the map display, as well as the text instructions.



#### 2.6.2.4) Favorites

All the screens of the “Favorite” menu are stored in classes of this package. This includes the screen of creation of a new favorite, edition of an existing favorite, deletion, etc.

The classes ListFavourite, DeleteFavourite, EditFavourite, NewFavourite are related to the screens of the favorite menu. ListFavourite displays the list of favorites, as well as the options available to the user (edit, delete, new, etc.). DeleteFavourite is a pop-up triggered when the user wants to delete a favorite. It asks him for a confirmation. EditFavourite is a screen that lets the user edit the address or name of one favorite. NewFavourite is the screen that enables the user to fill in information to create a new favorite.

#### 2.6.2.5) GPS

This package contains only one class: GPS. This class is responsible for the communication between the mobile phone and the GPS.

It works as an independent thread, which checks regularly the data given by the GPS. The retrieved data is used to locate the user, and get information on his speed, direction, etc.

#### 2.6.2.6) Map

An important part of the application is the display of the map. This is used by different parts of the application, like the simple map or the path display. This package is there to make the queries to Google maps, and display the result on the screen. Depending on which menu the user chose, a path will be displayed or not.

The Map package contains only the class Map. It is important to note that this class is related to two different screens. Indeed, the Map menu and the Destination menu both need to display the map. It was a programming choice to unite those two features in one class. Indeed, it seemed a good factorization of code.

This class contains the commands that can be found for both screens, as well as all the variables that are needed to draw points or paths on the map. A Boolean variable determines if the path and specific information should be displayed, or if it is just a display of the map.

#### 2.6.2.7) Menu

The first screen of our application is a menu which is displayed as an ellipsoid, and which rotates depending on the user’s will. This is a complex animation, which is entirely managed by this package. Depending on which icon the user selects, the program will show one of the other screens.

From a programming point of view, the application displays several icons, and plays an animation depending on which arrow is pressed by the user. Each time a rotation is finished, a new label is set in the front, to explain what the icon is representing. We will speak more about these animations in the “difficulties faced” part.

#### 2.6.2.8) Path

The mobile phone receives information from the server about the path the user should take in order to reach his destination. The mobile phone application must parse an XML file sent by the server, and which contains all the data of the path. To store this data, the application must also have classes defining the elements of this path (Nodes, Segments, Step, etc.).

#### 2.6.2.9) RecordManager

The mobile phone application does not store a lot of data. Except for the xml file (the path), the only information that the application must keep is the favorites and the settings. That information is set by the user, who can change them any time he wants to. For this reason, it seemed important to create a package to manage the records of this information. The package contains two classes: one to manage the favorite records, and the other to manage the changes in the settings.

#### 2.6.2.10) Settings

This package manages the settings screen. When the user wants to change the settings, this package will enable him to see the current settings, and to choose others, as well as save them in memory.

#### 2.6.2.11) Sound

This package manages all the sound playing in the application.

### 3) Server

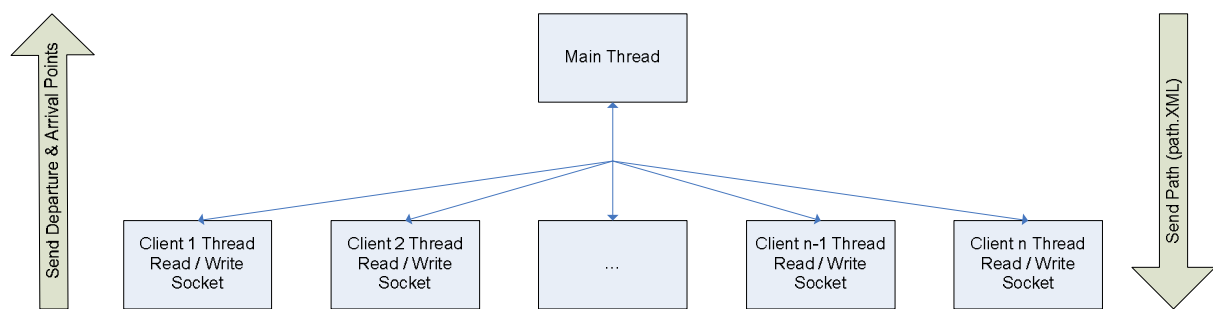
Regarding the programming language, our choice has mainly been conditioned by the fact that we restarted from the beginning a project made the year before, and which had been made in Java. We could have decided to change, but the portability of Java, allowing the use of the application server on different operating systems, convinced us not to do so.

Based on the analysis we made, we chose Windows XP Professional, since it is one of the simplest operating system to use, in the market, and with which we're more familiar. Another main reason was the fact that we had been given a computer already running Windows. As we could use Java on Windows, it has been our final configuration.

The server implementation is based on a classical multithreaded application schema. For each client device connected through the network to the server, the application creates a dedicated thread to handle efficiently the incoming/outgoing data.

The server includes the following functionalities:

- Interpreting "Open Street Map" data.
- Building a list of nodes, ways, bus lines, bus stops & connection nodes.
- Retrieving the NT Bus schedules.
- Computing a shortest path combining walking with bus.
- Creating/Sending a normalized XML file describing the path found.
- Communicating using TCP protocol on the internet with any type of client devices.



**Multithreaded Server Architecture**

We will study in the following parts the implementation of the different functionalities implemented in the server.

#### 3.1) Map

During the server design phase, one of the main issues was the shortest path calculation. Amongst the different solutions, we considered:

- Send a query to Google Maps and analyze the HTML response

- Download Aalborg geographic data and use Dijkstra algorithm to compute a path

### **Google Maps Advantages**

- Ready-made solution easily implementable using javascript and a java web crawler.

### **Google Maps Drawbacks**

- Modularity, we can't change the shortest path calculation settings.
- Any modification in the path description HTML/JavaScript code will induce a malfunctioning of the path web crawler.
- Complexity to bind the walking shortest path (Processed by Google Maps) with the bus shortest path (Processed by the Bus Server)

### **Own made solution Advantages**

- Modularity, we can change the shortest path calculation settings, add new data.
- Self sufficiency, all the geographic data are saved on the server.
- Walking and bus shortest path calculations can be integrated in a common method.
- An editable map on the website allows the integration of the NT Bus Stop positions.

### **Own made solution Drawbacks**

- The Map Provider, OpenStreetMap, is a collaborative project : anyone can modify the data and input inaccurate geographic data
- Aalborg map is incomplete and to update it requires a lot of work, and continuous maintenance.

### **Our choice**

Due to the lack of modularity of Google Maps path calculation method and the complexity to bind the walking shortest path we chose to develop our own cartography system using OpenStreetMap data. Using this solution enables us to use only one XML file to handle all the geographic data.

#### ***3.1.1) Data Processing***

In a first time, we will study the bus modeling, and then the streets modeling. Then we will have a look to the map class.

##### **3.1.1.1) Bus Line modeling**

Each bus line can be easily created and updated using an OpenStreetMap account and the map editor provided on the website. On the following picture, we are able to read the information describing each bus stop node (tag "name" for the name of the bus stop and "highway" for the node

type). A bus stop is then linked to a relation describing the bus line (tag "name" for the direction, "ref" for the number of the line and "route" to define the transportation mode)

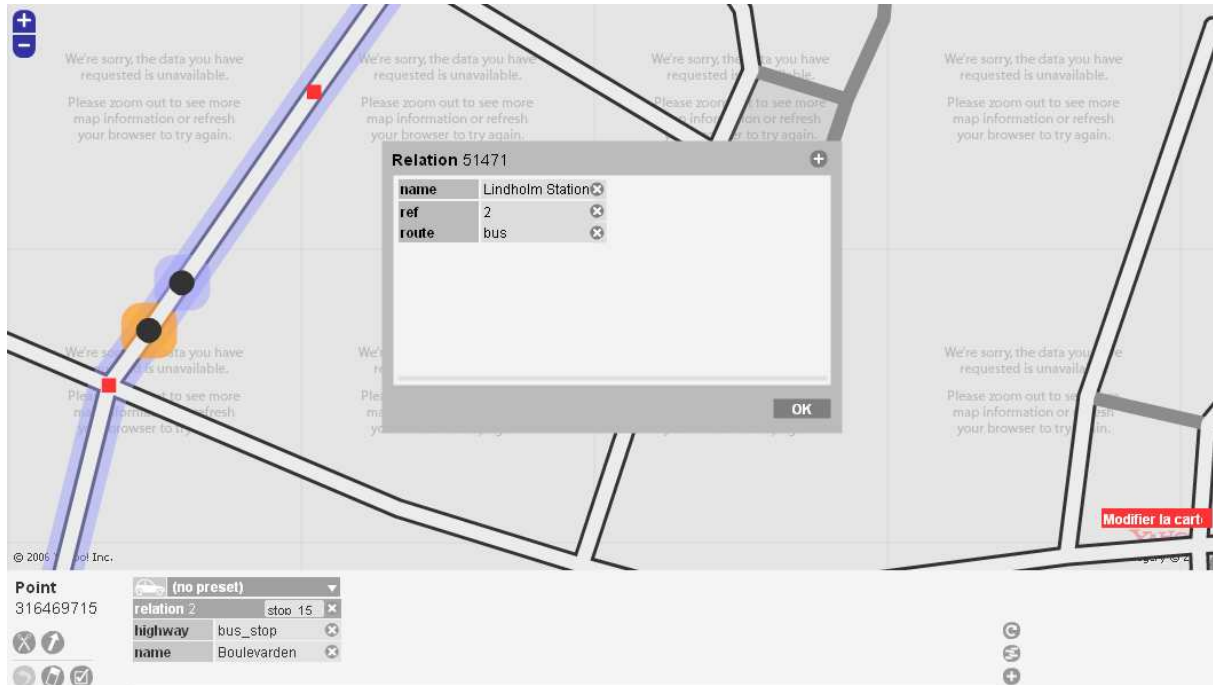


Figure 44 - Bus Line Creation

Once the bus line is created, we can download the data in a map.osm file using OpenStreetMap "Export" functionality. Let's look at the XML description of the information entered on the website.

```
<node id="316469715" lat="57.0457495" lon="9.9185768" user="Thomas Luel"
visible="true" timestamp="2008-12-03T17:55:59+00:00">
  <tag k="name" v="Boulevarden"/>
  <tag k="highway" v="bus_stop"/>
</node>
```

#### Bus Stop node Description

A Bus Stop is defined by a node id reference, latitude, longitude, highway type ("bus\_stop") and its name ("Boulevarden")

```
<relation id="51471" visible="true" timestamp="2008-12-03T17:55:59+00:00"
user="Thomas Luel">
  <member type="node" ref="27475672" role="stop_10"/>
  .....
  <member type="node" ref="316469715" role="stop_15"/>
  <tag k="ref" v="2"/>
  <tag k="created_by" v="Potlatch 0.10f"/>
  <tag k="name" v="Lindholm Station"/>
  <tag k="route" v="bus"/>
</relation>
```

### Bus Line relation Description

A Bus Line is defined by a relation id, a list of bus stops nodes and a list of tag ("ref", "name", "route") giving some information about the direction, the number of the bus. Each bus stop node is defined by a node "id" reference and its order ("stop\_15") in the bus stops list.

#### 3.1.1.2) Streets modeling

Using the same map.osm file as for the bus lines, we can also retrieve all the data describing the streets of Aalborg city.

```
<node id="29978688" lat="57.0469713" lon="9.9189735" user="ftmazzone"
visible="true" timestamp="2008-12-05T17:54:37+00:00" />
```

### Node Description

A node is described by its reference "id", latitude "lat" and longitude "lon".

```
<way id="8166939" visible="true" timestamp="2008-12-05T17:54:37+00:00"
user="ftmazzone">
  <nd ref="29978688" />
  <nd ref="316820788" />
  <tag k="oneway" v="yes" />
  <tag k="highway" v="residential" />
  <tag k="created_by" v="Potlatch 0.10f" />
  <tag k="name" v="Vingårdsgade" />
</way>
```

### Street Description

A street is described in a "way" element by its reference "id", its nodes "nd" and its "name". Retrieving this information is sufficient to create our own map and provide an efficient path to the user.

## 3.1.1.3) Map class – Data Parsing

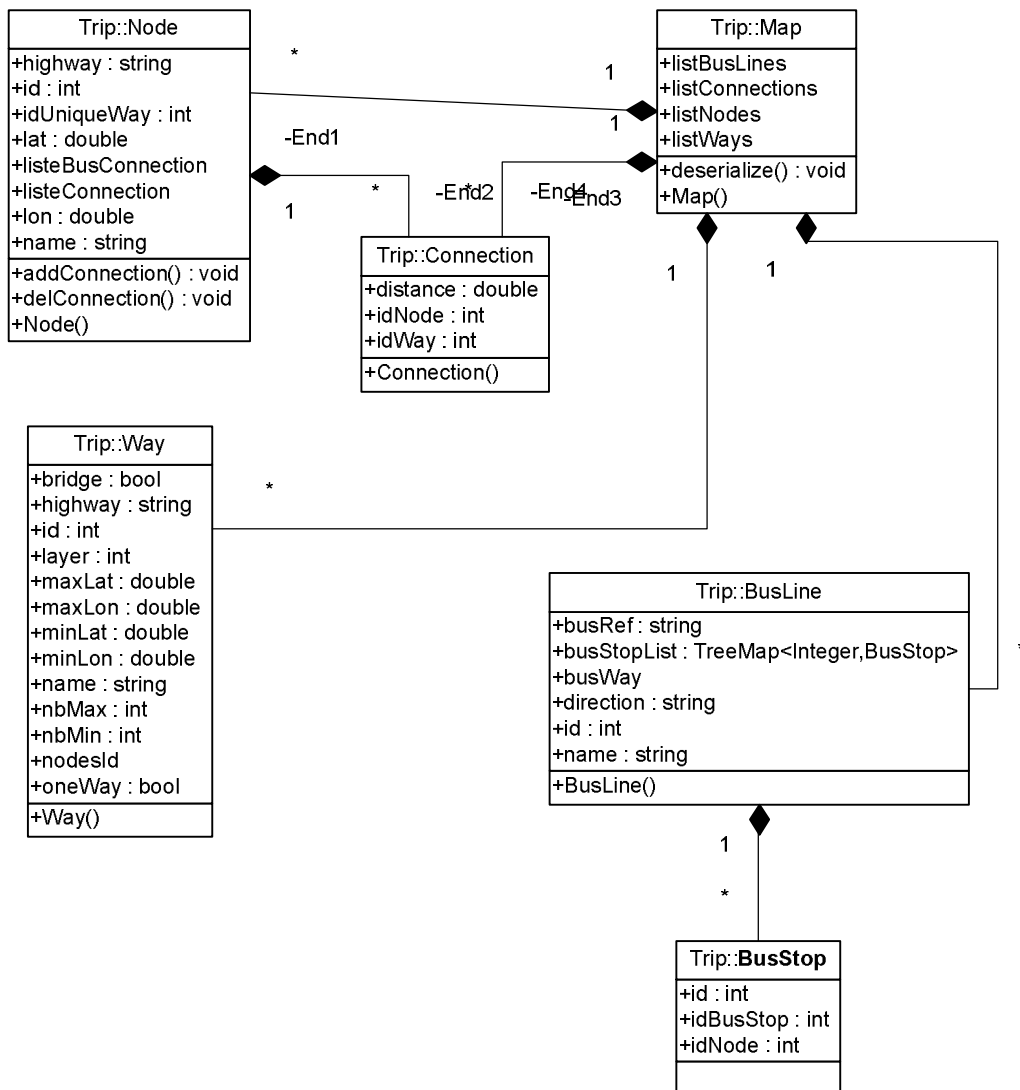


Figure 45 Map class diagram

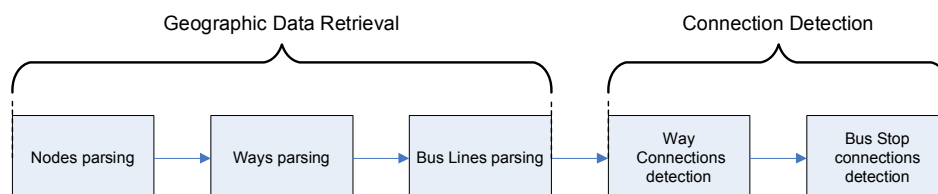


Figure 46 Information processing

During the server design, two solutions for the geographic data were studied:

- Create a relational database to save all the data and load only the needed information into memory (RAM)

- Parse the map.osm file at the starting up of the server and keep all the data into memory (RAM)

Using a database solution to retrieve the data would have enabled the server to be almost instantly ready at the startup but we finally chose to parse the map.osm file at each start up for multiple reasons:

- We can accept a startup duration of 200~300 seconds for a server application infrequently stopped.
- The Aalborg area map is updated every week, updating this data in the data would have required the development of a tool of maintenance/updating.
- The Aalborg area map.osm is a light xml file (~2 mega), our tests of data parsing with a java DOM<sup>1</sup> parser showed that it could parse much heavier files (at least 10 mega). Moreover the size in RAM memory of the java objects storing the geographic data was insignificant for the server (Computer RAM = 2 Giga, java test software ~ 30 mega).
- The data access time while calculating a shortest path is insignificant when the data is stored in the RAM memory.

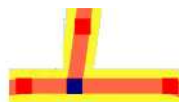
We designed the class map to parse the XML file using a DOM parser.

```
DocumentBuilderFactory docBuilderFactory =  
DocumentBuilderFactory.newInstance();  
  
DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();  
Document doc = docBuilder.parse(new File(addressFile));  
doc.getDocumentElement().normalize();
```

#### Dom parsing

Each data (street, node, bus stops and bus lines) read by the parser is used to create an object (Node, Way, BusLine, BusStop ) stored in object lists (TreeMap lists) as shown in the previous class diagram.

#### 3.1.1.4) Map class – Connections processing



---

<sup>1</sup> DOM parser : Document Object Model



The streets and bus lines connections list is a pre-requisite for a shortest path calculation method. Since the map.osm file retrieved on OpenStreetMap website does not provide it, we designed an algorithm to create this list.

As explained in the Data Parsing part, a street is defined by a list of nodes describing its shape, each street having at least two nodes. Two streets are connected if and only if they have a common node in their nodes list. Regarding the bus line, each bus stop node is connected to others bus stops nodes (at least one).

- *Connections handling*

Two solutions to handle the connections were studied during the design period:

- Alter the "Way" object and add a list of connections with others streets
- Alter the "Node" object and add a list of connections with others nodes

We finally chose to implement the connections list in the "Node" object. As shortest paths algorithms use nodes to find a path and not ways, using this solution was much simpler for the implementation and avoid using the "Way" objects during the calculation.

### Street Connection

Each "Node" object contains a list of connections to other nodes. Each connection is defined in a "Connection" object that contains :

- "idNode" a reference to the connected node.
- "idWay" a reference to the way connecting the two nodes.
- "distance" the distance between the two nodes.

### Bus Stops Connection

Each "Node" object contains a list of connections to the next bus stops of the lines serving this stop. A connection is defined using a "BusStop" object that contains :

- "id" a reference to the bus line id concatenated with the order number of the following bus stop.
- "idBusStop" the order number of the following bus stop.
- "idNode" a reference to the node id of the following bus stop.

**Remarks:** This design is totally scalable, to add a new transportation mode (train & tramway...), the developer can easily develop new objects (with a similar design to the existing ones) and integrate them in the existing code.

- *Connection discovery*

We designed an algorithm to retrieve the connection between the nodes using a set of rules. To be considered as a connection, a node has to:

- Be one of the edge nodes of a way (first or last node) OR/AND
- Be in common in the nodes lists of at least two ways OR/AND
- Be a bus stop

To find these nodes, we execute the following algorithm:

For each way //Initialization

distance = 0 //Initialize a distance variable to 0

refConnexA = refNode

refConnexB = 0

//Initialize the previous connection variable to the first node id of the way

//Initialize the current connection variable value to 0

From the second to the second last nodes (N) of the current way

distance = distance + distance between the node (N) and node (N-1)

Check if the node (N) is a bus stop

Check if the node (N) is used in another way nodes list

If one of the above conditions is true

refConnexB = refNode (N)

//Copy the reference of the current node to the refConnexB

listNode[refConnexA].Add(Connexion(refConnexB))

listNode[refConnexB].Add(Connexion(refConnexA))

//Update the nodes list and add crossed references between two nodes including the way identifier and the distance between the two nodes

distance = 0 //Reset the connexion distance to 0

refConnexA = refConnexB

//Increment the value of the previous connection

End if

End loop

refConnexB = refNode

//Copy the reference of the last node id of the way

listNode[refConnexA].Add(Connexion(refConnexB))

listNode[refConnexB].Add(Connexion(refConnexA))

//Update the nodes list and add crossed references between two nodes including the way identifier and the distance between the two nodes

End loop

//Retrieval of the bus stop connections

For each bus line

busStopA = identifier of the first bus stop of the line //Initialize the previous bus stop variable

From the first to the second last bus stop of the current bus line

add to the busStopA a reference to the next bus stop

End loop

End loop

### Example of Connections discovery:

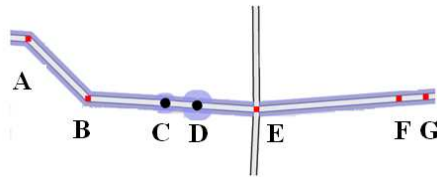
The connection studied is colored in yellow.

A connection found is colored in blue.

A black node is a bus stop.

A red node is a node without any connection (or not studied).

### Initialization



### Step 1 – First Node of the way

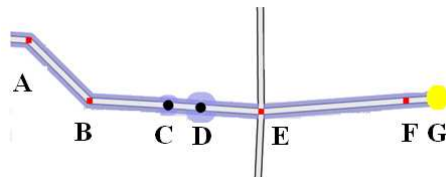
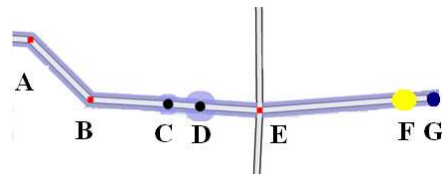


Figure 47 Way to study

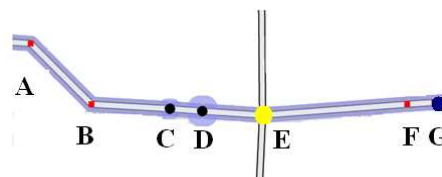
First node of the way → it is a connection.

### Step 2

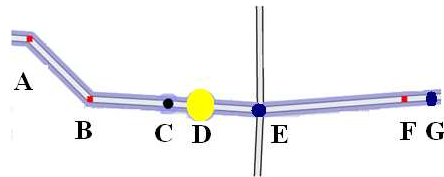


This node is specific to the current way → it isn't a connection.

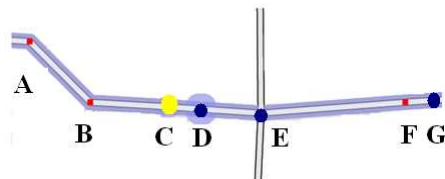
### Step 3



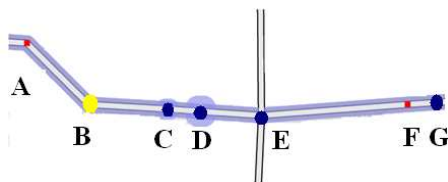
Two ways share this node → it is a connection.

**Step 4**

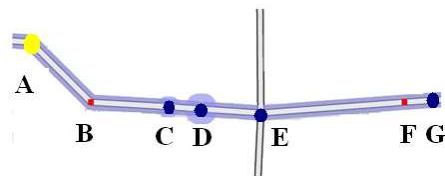
The node is a bus stop → it is a connection.

**Step 5**

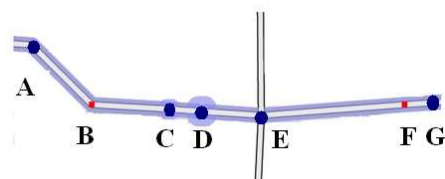
The node is a bus stop → it is a connection

**Step 6**

This node is specific to the current way → it isn't a connection.

**Step 7**

The node is the last of the current way → it is a connection

**Last Step**

### 3.2) Geocoding

(Wikipedia s.d.) "Geocoding is the process of finding associated geographic coordinates (often expressed as latitude and longitude) from other geographic data, such as street addresses, or zip codes(postal codes)". In order to calculate a path, we need to locate the departure point and the arrival point in the system we are using. As we have said previously in his report, we decided to use OpenStreetMap as a map for the path calculation. This means we are using the nodes and the segments of OpenStreetMap to compute the Dijkstra algorithm.

This implies that we need to determine where the user starts and where is his destination in OpenStreetMap. That means we need to translate the GPS coordinates or the address we get into a Node identifier in OpenStreetMap. For the destination, we will always get an address (given by the user). But for the departure point, it will depend if the user checked the box "start from my current position". If he checked the box, then we will get the GPS coordinates of his current position. If he did not, he will have typed an address in the form.

There are thus two parts to consider. Indeed, if we get an address, we need to:

- Convert the address to GPS coordinates.
- Use the GPS coordinates to find the nearest node in OpenStreetMap.

If we get GPS coordinates instead of an address, then we only execute the second step. We will now explain the two steps with more details.

#### 3.2.1) *Convert the address to GPS coordinates*

Several choices were available to us to make this conversion. There were indeed several website that could help us with the geocoding. We found for example "TravelGis", which locate any address you give. However, OpenStreetMap and Google API also offered the geocoding service. Since we were already using those two services, we decided that it would be more efficient to use them for this conversion from address to coordinates.

After a quick study, we decided that Google API offered the quicker way to do what we wanted to do. There is indeed a way to make a HTTP query to Google API, giving it an address and getting the latitude and longitude associated to it. HTTP query being easy and efficient, we decided that it would be a good way to locate the user. Google API documentation gave us the information about how to build such a query and retrieve a XML file.

The queries are built this way:

*`http://maps.google.com/maps/geo?q="name of the street",+"name of the city",+"name of the country"&output=xml&oe=utf-8&key="google API key"`*

The output parameter is set at "xml" because it is the easiest way to retrieve information from this kind of request. The "oe" parameter defines the encoding of the XML file which is returned by the query. We set it on "utf-8" because the encoding works with the Danish special letters (æ, ø,

å). Without this setting, the XML file would not be encoded in UTF-8, and that would create an error during the parsing. Finally, the parameter “key” takes our google API key.

An example of a query could be:

<http://maps.google.com/maps/geo?q=Vesterbro,+Aalborg,+Denmark&output=xml&oe=utf-8>

This HTTP query will return this XML file:

---

```
<?xml version="1.0" encoding="UTF-8" ?>
<kml xmlns="http://earth.google.com/kml/2.0"><Response>
  <name>Vesterbro, Aalborg, Denmark</name>
  <Status>
    <code>200</code>
    <request>geocode</request>
  </Status>
  <Placemark id="p1">
    <address>Vesterbro, 9000, Danemark</address>
    <AddressDetails Accuracy="6" xmlns="urn:oasis:names:tc:ciq:xsd:schema:xAL:2.0">
      <Country>
        <CountryNameCode>DK</CountryNameCode>
        <CountryName>Danemark</CountryName>
        <AdministrativeArea>
          <AdministrativeAreaName>Jutland du Nord</AdministrativeAreaName>
          <Locality>
            <LocalityName>Aalborg</LocalityName>
            <Thoroughfare>
              <ThoroughfareName>Vesterbro</ThoroughfareName>
            </Thoroughfare>
            <PostalCode>
              <PostalCodeNumber>9000</PostalCodeNumber>
            </PostalCode>
          </Locality>
        </AdministrativeArea>
      </Country>
    </AddressDetails>
    <Point><coordinates>9.9141145,57.0468014,0</coordinates></Point>
  </Placemark>
</Response></kml>
```

---

In this XML, you can see a lot of information about the location found by Google maps. We don't need most of this information. Actually, the only important line is the one that appears in bold in this report. You can see the mark-up “Point” containing the mark-up “coordinates”. In this last mark-up, we can find the longitude and the latitude of the point located by google maps.

To get those values into our program, we use an XML parser called KXML (a SAX parser). With this parser, we locate the mark-up "Point", and then the mark-up "coordinates", and we retrieve the content.

### 3.2.2) Find the nearest node in OpenStreetMap

The OpenStreetMap map of Aalborg is stored in a XML file. It is required to compute the dijkstra algorithm. At the start of the server application, it will copy all its content into the memory. This will also be useful to find the nearest node to a GPS position.

First, it is important to check if the position we have is within the map boundaries. Indeed, our project only works within a area defined before we launch the server. During our tests, this area

contained Aalborg and its surroundings. Since the position we get could be anywhere on earth, we need to check first if it can be handled by our program. After this quick check, we can begin to search for the nearest node.

We had several choices of algorithms to get the ID of the nearest node. We will explain briefly each of them, going from the most simple to the most complex. In order to explain in a good way, we will name the position "Position U", since most of the time it is related to the User (or given by him).

- Browse the Node list and calculate the distances from one Node to the position.

This is the simplest algorithm. We thought about browsing the list of nodes, and finding out which one has the shortest distance to the position U. However, during our tests, there were more than 10 000 nodes in the area we were working with. This algorithm would thus require a lot of computation. We decided that it was not efficient, and that it could be improved.

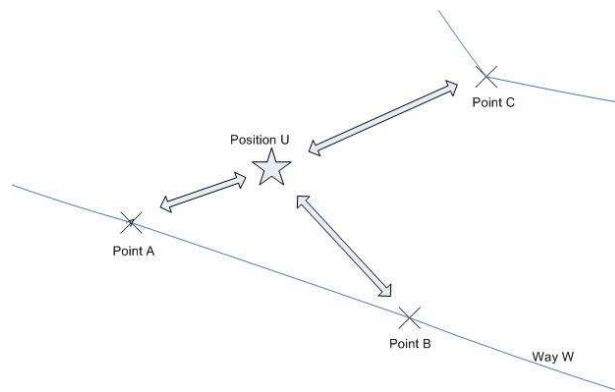


Figure 48 : Calculation of the distance between U and A, B, C.

On the diagram, you can see the position U represented by a star. There are also three points on two ways. The algorithm would calculate the distances between U and A, U and B, and U and C. Since A is the nearest, it would be chosen by the algorithm. Note that the position U is not on the way W, because of an approximation due to the GPS system. Indeed, the GPS system is not very precise, and we have to deal with this problem in our program. We will speak later about the issues due to the GPS imprecision.

- Select only the surroundings ways

We needed to reduce the list of nodes used in the distance calculation. The best way to do that was to calculate the distance to the position U only for the nodes belonging to some selected ways. But which way do we select? After some thinking, we decided we could select only the ways that went near to the position U. We thought about delimiting a square around the position U, and select all the ways going inside this square.

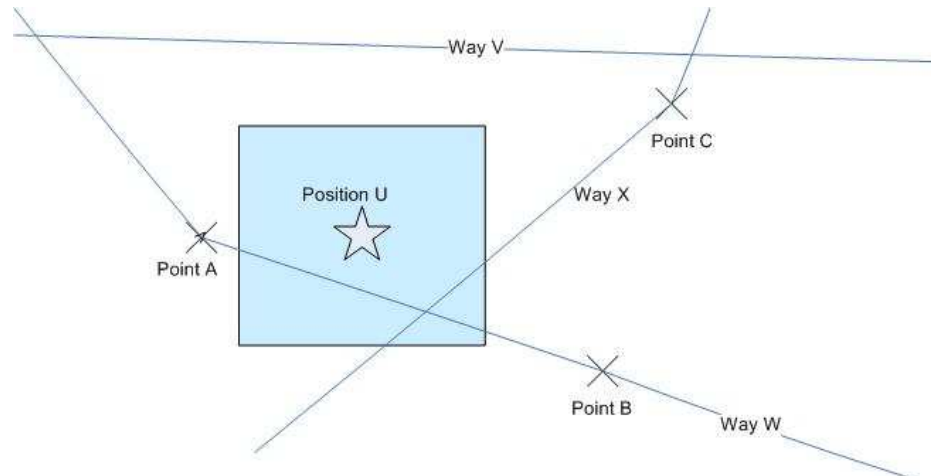


Figure 49 : The ways W and X will be selected.

The figure above gives an example. There are three ways coming near the position U. The algorithm delimits a squared area around the position U, and check, for each way, if they go inside the square. Thus, the ways W and X will be selected, and not the way V. After this selection, the algorithm would browse the lists of nodes of each way, and calculate the distance to the position U, exactly like the first algorithm described previously.

- Find the nearest segments

We decided to implement the previous algorithm, and it worked well. But there was a specific case when the result was not satisfactory. In order to fully understand the issue we faced, there are a few points that must be clarified.

First, it is important to note that the GPS system is not entirely precise. We have said it already. Also, there is a notable difference between OpenStreetMap and Google maps coordinates. That means that the position we retrieve may be quite far from any ways on OpenStreetMap. For this reason, in some cases the position U was quite far from the way it was supposed to be on. The following figure explains the issue in a better way.



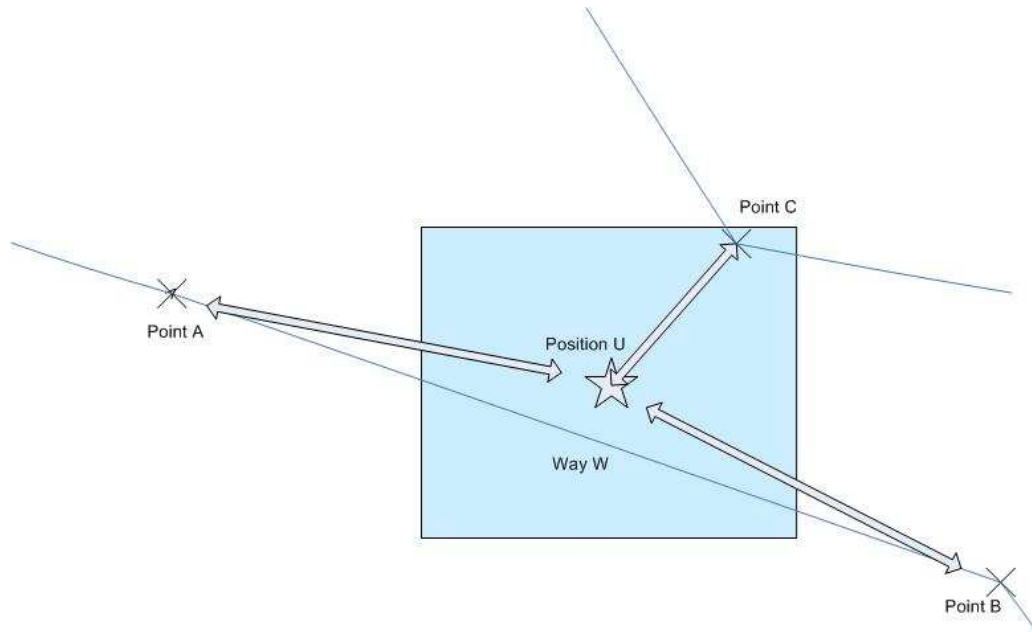


Figure 50 : The point C is the nearest, but the way W is nearer.

On the figure above, we see that the position U is quite near to the way W. There is a high probability that the position U “should be” on the way W. However, with the algorithm we had designed, the point C being the nearest of all, it was chosen as the nearest point. The position U was thus associated to the wrong way, which could be a big problem for the application.

In order to solve this issue, we came up with a slight change in the algorithm. We decided to keep the squared area filter to select the ways. But then we thought that it would be better to first select the nearest segment, rather than the nearest node. This way we can make sure that the selected node will be on the right way. The segment is the link between two nodes.

To calculate the distance between the position and the segment, we first determine the intersection between the segment itself and the perpendicular line passing by U. We name this intersection the point P. If the point P is between the two extremities of the segment, then we consider the length of the perpendicular line as the distance between U and the segment. If the point P is not between the two extremities of the segment, then we consider the distance to the nearest of the two extremities as the distance to the segment. The following figures complete this explanation.

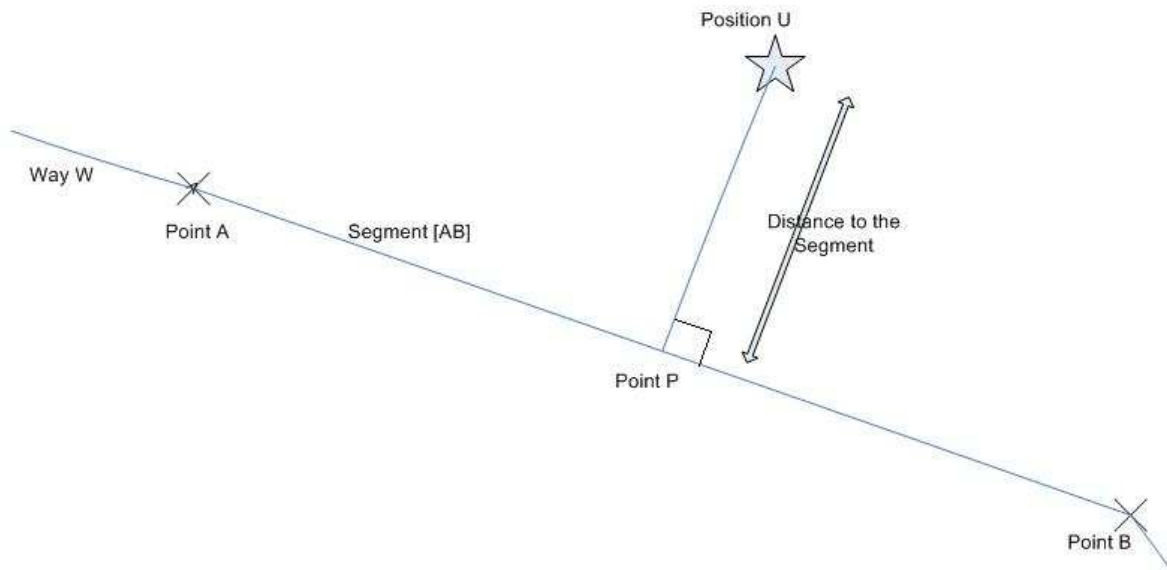


Figure 51 : the distance to the segment is the distance to the point P.

Once the nearest segment has been found, we choose the nearest extremity as the nearest node. Let us sum up the algorithm:

1. Select the ways that goes inside a squared area placed around the position U.
2. Browse all the segments of those selected ways, and determine the nearest one, by calculating the distance between the position U and the segments (see above for the method we used).
3. Calculate the distance to the two extremities of the nearest segment, and select the nearest of the two.

If we take the same case as before, the selected node would be the point A or B, but not C.

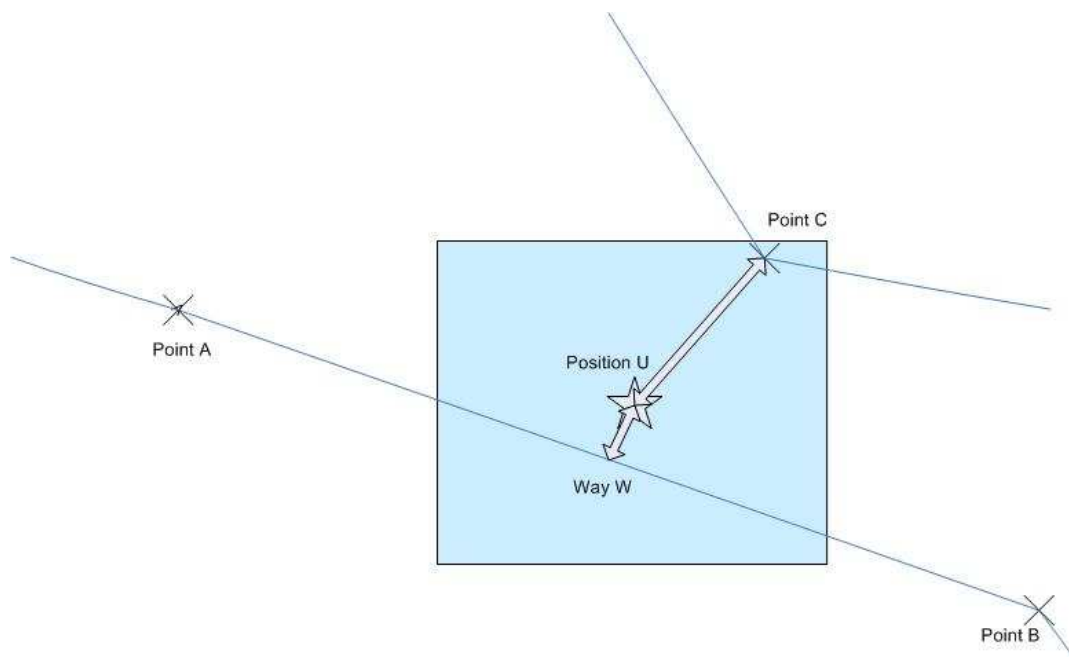


Figure 52 : The Segment [AB] is selected. The nearest point will be the nearest between A and B

This last algorithm was the one we chose, because it better suited our needs.

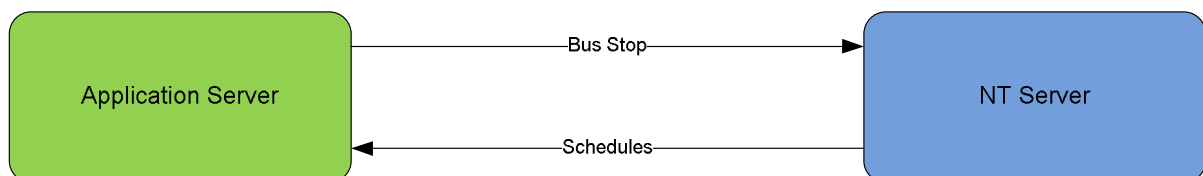
### 3.3) Bus schedule retriever

Although all the calculation made in the main application in the server, it is also necessary to think about how will be the communication between the application server and NT. Unfortunately, we didn't have access to NT Server, and so the application server has to retrieve the information about the bus schedule directly from the NT's website. Actually, this was a problem since the beginning of the design, and all the things in design and implementation had to be rethought. We'll talk in this section about the first idea for the communication between the server and NT, and also the final design, made knowing that it will be impossible to take the information directly from the NT's server.

#### 3.3.1) The first design

In the beginning of the development of this block, we thought to retrieve directly the information from the NT's server. After that, it should be easy to take the time of a bus. In fact, when we started to think about this, we thought to include this block in the main program, since it will not be necessary too much processing, in the case that the server communicates directly with NT's server.

Thus, the main idea of this first design is described in the following diagram:

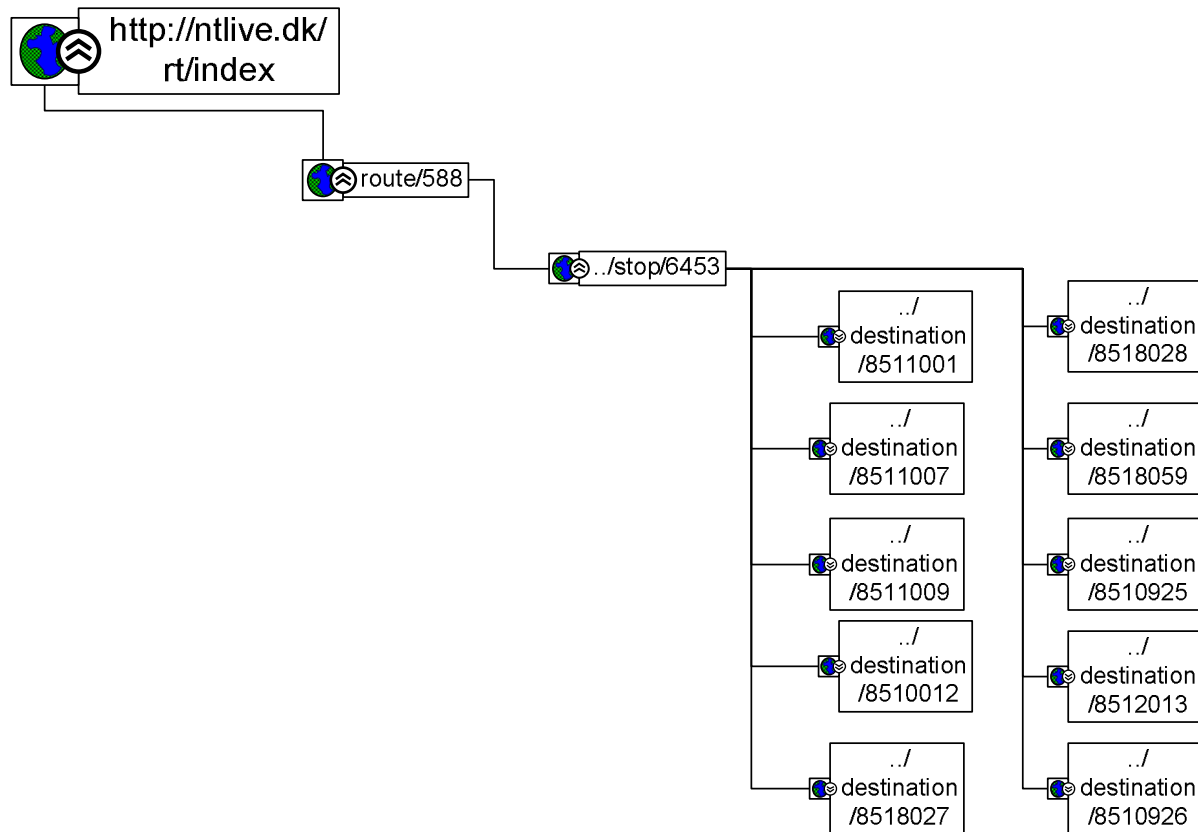


1 – In the first step, our application server simply will send the name or the id of the bus stop.

2 – The NT Server with the name or the id of the bus stop, it will send back to our server the schedule for that bus stop. After that, the application server will do all the necessary processing to take the best bus for the user's path.

As described previously, unfortunately this solution couldn't be applied. We tried to contact NT for several times, but always without answer. So, we had to start thinking about another solution to take the bus schedule. Since NT has a website with such information, we thought to take directly the information from NT's website.

In the following picture, is described how is organized the NT's website, from the root <http://ntlive.dk/rt/index>:



In the root page, there are all the routes from NT. In our case, it is only interesting the route 588 (Aalborg). After this, we enter in the page of all the bus lines. Just for example, we selected the stop 6453 (route 74). Finally, we have the list of all the bus stops of that bus line. Just one more click in one of the bus stops, and we have the schedule in that bus stop. With this in consideration, we designed again the communication between the server and NT.

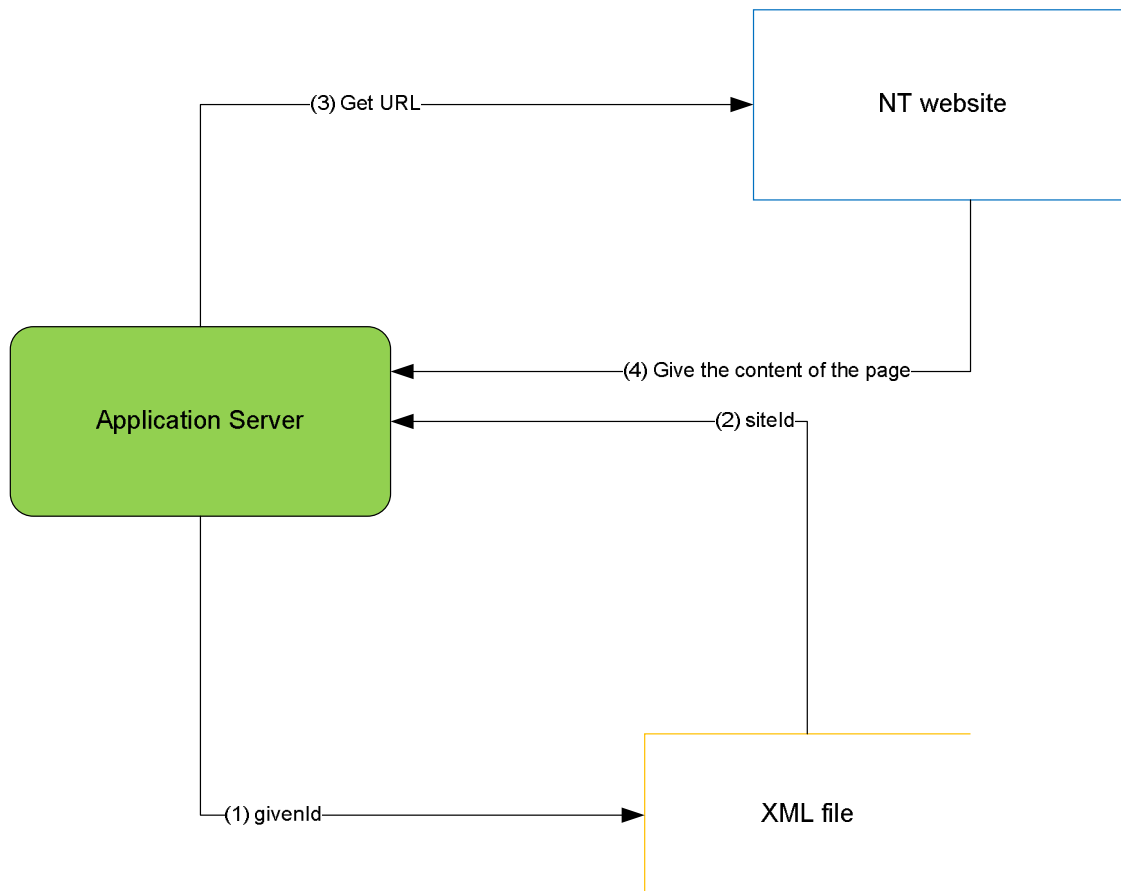
### 3.3.2) Getting information from NT's website

Since a bus stop is identified by a reference number in open street maps, and also by a reference number in the website (the last number of the path), we had to associate these 2 numbers for each bus stop in a XML file. Each line of this XML file will be like is shown [Appendix 2]:

```
<busStop relationId="51040" givenId="51040000" siteId="8510024" />
```

- *relationId*: this field represents a bus line (2, 12, 14, ...) for a specific direction (Aalborg, University, ...). For example, for the bus number 2 in the direction of University, the *relationId* should be 51040. This number is auto generated in open street maps, when is created a bus line.
- *givenId*: is the *relationId* plus the number of the bus stop in that bus line. This number starts in "000" in the first bus stop, and goes on in an increasing order. For example, in the direction of University, the first bus stop should have the *givenId* like 51040000.
- *siteId*: is the id of this bus stop in the NT website. In the end, the URL in which the information will be retrieved should be: <http://ntlive.dk/rt/destination/> + *siteId*.

So, in a first look, the communication between the server and NT will be like is shown in the following figure:



The numbers in the picture give the order of the events in the communication between both parts. So for a given id of the bus stop, the first step is to check the right *site id*, which means, the final part of the page's URL, as described previously. Thus, the next step is to get the information from the webpage, and treat the information given by the website.

Finally, the information given by the NT's website is in a HTML format, like the following:

```

<h3>Aalborg Busterminal - 16:59</h3>
<div><hr/></div>
<ul style="margin: 0;">
<li>2H mod Klarup<br/>
... 17.00
+2</li>
<li>17 mod Saltumvej<br/>
... 17.00
+3</li>
<li>13 mod Fjordparken<br/>
... 17.00
* </li>
<li>13 mod Gug Øst<br/>
... 17.00
* </li>
<li>2B mod Nørhøne/Airport<br/>
... 17.00
</li>
<li>17 mod Strubjerg<br/>
... 17.00
</li>
<li>12 mod Vesterkøret<br/>
... 17.08
</li>
<li>12 mod AAU Busterminal<br/>
... 17.08
</li>
<li>2C mod Uttrup Nord<br/>
... 17.08
</li>
<li>2K mod Gistrup/Skolen<br/>
... 17.08
</li>
</ul>
<p class="text">* = <em>Køreplankid</em></p>
<p><a href="http://ntlive.dk/rt/destination/8510027?offset=10">Se flere afgange</a></p><div><hr/></div>
<div>

```

### 3.3.3) Parsing the information

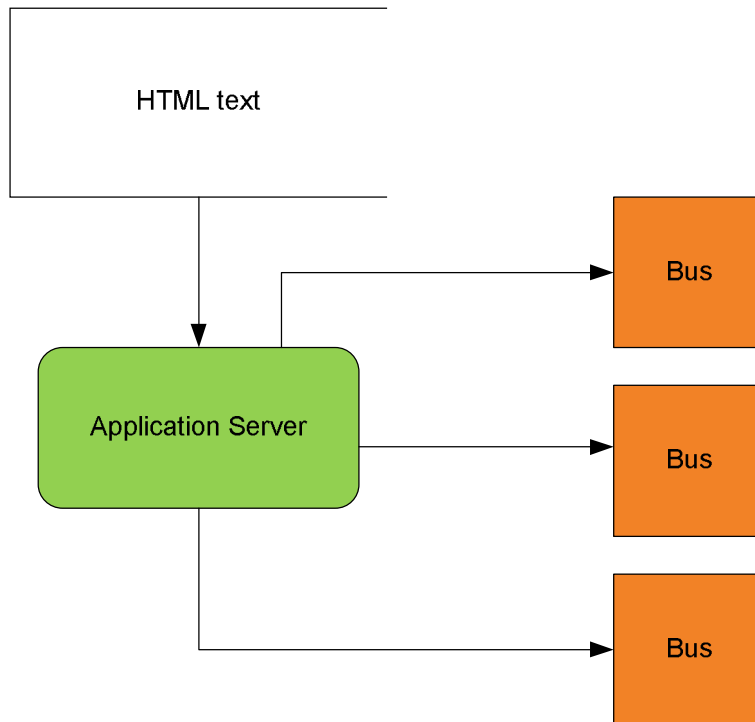
In the previous code, the relevant information is between the `<li>` blocks. So first of all, it's necessary to do the parsing of this information. For example, in the first `<li>` block, we should have:

*2H mod Klarup <br/> ... 17.00*

Thus, it is important to parse this information to take the following information:

- Bus number: in this example, should be 2H.
- Destination: in this example, should be Klarup.
- Time of departure: in this example, should be 17.00.
- Delay: in this example there's no delay, so should be 0.

After parse all of this information, we converted the text from the web page to an object. In this case to an object *Bus*.



The class Bus is shown in the following picture:

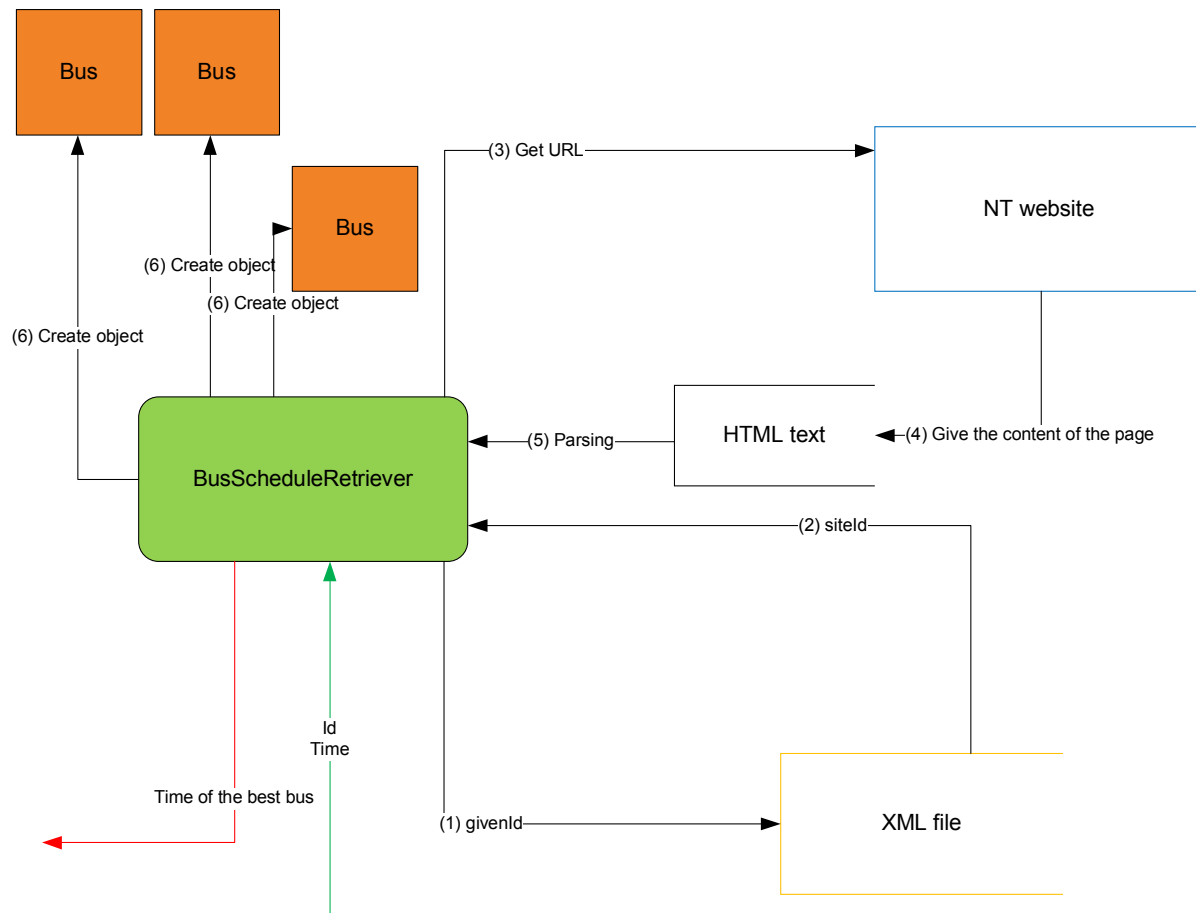
Bus
-_number : String
-_destination : String
-_hour : Integer
-_minute : Integer

### 3.3.4) Selecting the bus

The real purpose of all of this processing is to give in the end the time of the first bus that arrive the bus stop after the user arrives. For this, since we have all the busses in a list, it is time to select the best bus to return to the main program.

If the number and the destination correspond to one of the numbers and destinations in the list, we just check if the time of the bus is above of the time that is passed to the program. For instance, those lists of numbers and destinations are lists that we choose for the program. We had to choose such numbers and destinations, because in Aalborg there are a big quantity of busses and destinations, and to retrieve all of this information will be very heavy for the server. We decided to choose only the bus line number 2 in both directions (South and North).

In conclusion, the following diagram explains all the procedure of this block of the server:



### 3.4) Shortest path calculation

Once the departure and arrival nodes have been found, the server calculates the shortest path between these two points. In this part, we will describe how this task is computed by the server.

#### *IV.B.1.a) Tool case*

We will use the loxodrome (spherical earth model) method to calculate a precise distance in meters between two points defined by latitude and longitude and to determine the heading between two ways. [xvii - James Alexander, 2004]

#### *IV.B.1.b) Shortest Path algorithm*

During the analysis phase, we read and analyzed many documents related to “shortest path computation” but none of them were dealing with our specific need: Combine two means of transportation in a unique method.

For this, we created from scratch a specific algorithm based on Dijkstra’s method using multiple previous node references depending on the mean of transportation used.



```

//Initialization of the list of nodes to study
//Set all the nodes to "Unread Node State"
For Each Node in the node list
    Initialize the "Previous Node" variable to -2
    Initialize the "Previous Bus Stop" variable to -1
    Initialize the "Arrival Date" to null
End loop

//Launch the algorithm: set the departure node to studied node
Set the "Departure Node Arrival Time" to departure date
Initialize the "Studied Node" to the "Departure Node"

//Search loop
While there are some unstudied node and that a "Studied Node" is found

    //Update the arrival time for the connected node (by walk) to the studied node
    For each node ("A") connected to the "Studied Node"
        Calculate the "Walk Time" between the "Studied Node" and "A"

        //Update the arrival time and previous node variables if a better path by walk
        to go to the connected node is found
        If "Studied Node" Arrival Time + "Walk Time" < "A" Arrival Time
            Set the "A" Arrival Time to "Studied Node Arrival Time" + "Walk Time"
            Set the "A" Previous Node to "Studied Node" reference
            Set the "A" Previous Bus Stop Node to -1
        End if
    End loop

    If the "Studied Node" is a bus stop
        //Update the arrival time for the connected node (by walk) to the studied node
        For each bus stops node ("B") connected to the "Studied Node"
            Set "C" date variable to the arrival time at B

            //Update the arrival time, the previous node and the previous bus
            stop variables if a better path by bus to go to the connected node is found
            If "C" < current arrival time at B
                Set the "B" Arrival Time to "C"
                Set the "B" Previous Node to "Studied Node" reference
                Set the "B" Previous Bus Stop Node to "Studied Node" bus stop reference
            End if
        End Loop
    End if

    //Determine amongst the nodes the next studied node
    Set "Studied Node" Arrival Time to -1

    For each node ("D") of the nodes list
        If "D" has not already been a "Studied Node"
            If "Studied Node" Arrival Time is equal to -1
                Set "D" to the "Studied Node"
            End if
        End if
    End loop
  
```

```

Else if "D" Arrival Time < "Studied Node" Arrival Time
    Set "D" to the "Studied Node"
End if
End if
End Loop
End Loop

```

#### IV.B.1.c) Example of Shortest Path Calculation

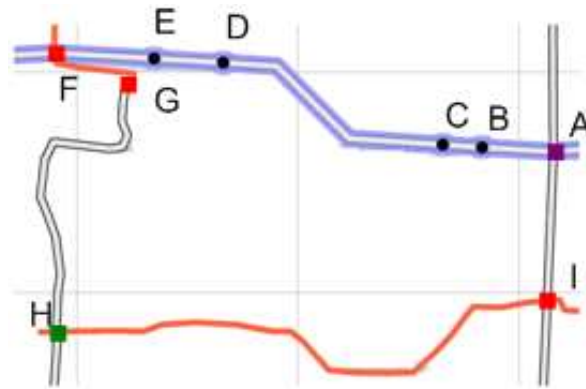


Figure 53 Map analyzed by the algorithm

**Departure dot:** violet rectangle  
**Arrival dot:** green rectangle

**Node:** red rectangle  
**Bus stop:** black circle

List of connections:

Node	Connection	Cost (in minute)
A	B	2
B	A	2
A	I	6
I	A	6
B	C	1
C	B	1
C	D	8
D	C	8
D	E	1
E	D	1
E	F	2
F	E	2
F	G	1
G	F	1
G	H	7
H	G	7
H	I	15
I	H	15

**List of bus stops connections:**

Node	Connection	Departure Time	Cost (in minute)
B	D	12:05	2
E	C	12:10	2

**Task:** Find the shortest path between A and H

**Step 1 – Studied Node = A**

Departure Time: 12:00

We set the studied node to the departure node A. B and I are connected to "A". "B" and "I" are connected to A.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	-2	-2	-2	-2	-2	-2	A
Previous Bus Stop	-1	-1	-1	-1	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	null	null	null	null	null	12:06

Amongst the unstudied node, "B" is the node with the "lowest" arrival time. "B" becomes the studied node.

**Step 2 – Studied Node = B**

Departure Time: 12:02

"C" is connected to "B" by walk and "D" is connected to "B" by bus.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	B	B	-2	-2	-2	-2	A
Previous Bus Stop	-1	-1	-1	B	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	12:07	null	null	null	null	12:06
2	12:00	12:02	12:03	12:07	null	null	null	null	12:06

Amongst the unstudied node, "C" is the node with the "lowest" arrival time. "C" becomes the studied node.

**Step 3 – Studied Node = C**

Departure Time: 12:03

“D” is connected to “C”.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	B	B	-2	-2	-2	-2	A
Previous Bus Stop	-1	-1	-1	B	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	12:07	null	null	null	null	12:06
2		12:02	12:03	12:07	null	null	null	null	12:06
3			12:03	12:07	null	null	null	null	12:06

Amongst the unstudied node, “I” is the node with the “lowest” arrival time. “I” becomes the studied node.

**Step 4 – Studied Node = I**

Departure Time: 12:06

“I” is connected to “H”.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	B	B	-2	-2	-2	I	A
Previous Bus Stop	-1	-1	-1	B	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	12:07	null	null	null	null	12:06
2		12:02	12:03	12:07	null	null	null	null	12:06
3			12:03	12:07	null	null	null	null	12:06
4				12:07	null	null	null	12:21	12:06

Amongst the unstudied node, “D” is the node with the “lowest” arrival time. “D” becomes the studied node.

**Step 5 – Studied Node = D**

Departure Time: 12:07

“D” is connected to “E”.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	B	B	D	-2	-2	-2	A
Previous Bus Stop	-1	-1	-1	B	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	12:07	null	null	null	null	12:06
2		12:02	12:03	12:07	null	null	null	null	12:06
3			12:03	12:07	null	null	null	null	12:06
4				12:07	null	null	null	12:21	12:06
5				12:07	12:08	null	null	12:21	

Amongst the unstudied node, “E” is the node with the “lowest” arrival time. “E” becomes the studied node.

**Step 6 – Studied Node = E**

Departure Time: 12:08

“F” is connected to “E”.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	B	B	D	E	-2	I	A
Previous Bus Stop	-1	-1	-1	B	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	12:07	null	null	null	null	12:06
2		12:02	12:03	12:07	null	null	null	null	12:06
3			12:03	12:07	null	null	null	null	12:06
4				12:07	null	null	null	12:21	12:06
5				12:07	12:08	null	null	12:21	
6						12:10	null	12:21	

Amongst the unstudied node, “F” is the node with the “lowest” arrival time. “F” becomes the studied node.

**Step 7 – Studied Node = F**

Departure Time: 12:08

“G” is connected to “F”.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	B	B	D	E	F	I	A
Previous Bus Stop	-1	-1	-1	B	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	12:07	null	null	null	null	12:06
2		12:02	12:03	12:07	null	null	null	null	12:06
3			12:03	12:07	null	null	null	null	12:06
4				12:07	null	null	null	12:21	12:06
5				12:07	12:08	null	null	12:21	
6						12:10	null	12:21	
7						12:10	12:11	12:21	

Amongst the unstudied node, “G” is the node with the “lowest” arrival time. “G” becomes the studied node.

**Step 8 – Studied Node = G**

Departure Time: 12:08

“H” is connected to “G” and it is faster to go to “H” from “G” than from “I”.

Node	A	B	C	D	E	F	G	H	I
Previous Node	-2	A	B	B	D	E	F	G	A
Previous Bus Stop	-1	-1	-1	B	-1	-1	-1	-1	-1
Step	A	B	C	D	E	F	G	H	I
1	12:00	12:02	null	12:07	null	null	null	null	12:06
2		12:02	12:03	12:07	null	null	null	null	12:06
3			12:03	12:07	null	null	null	null	12:06
4				12:07	null	null	null	12:21	12:06
5				12:07	12:08	null	null	12:21	
6						12:10	12:11	12:21	
7						12:10	12:11	12:21	
8								12:18	

Amongst the unstudied node, “H” is the node with the “lowest” arrival time. “H” becomes the studied node. As all the nodes have been studied, it means that the shortest path has been found. We are now able to describe the path to follow.

## Conclusion

We now are able to describe the shortest path by simply enumerate the previous nodes. As “D” node has a previous bus stop, we know that we have to take the bus from “B” to “D” and to walk between the others nodes.

Path:  $A \rightarrow B \rightarrow D \text{ (by bus)} \rightarrow E \rightarrow F \rightarrow G \rightarrow H$

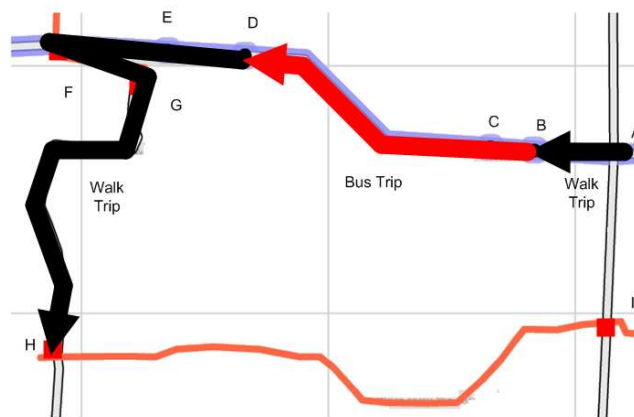


Figure 54 Shortest Path on the map

Once that the shortest path has been calculated, the server sends the path to a XML file generator method (“WritePathXml”) that write in a XML document sent to the mobile phone.

### IV.B.1.a) Xml file generator (“WritePathXml”)

As the server is going to send the shortest path calculated through the network, we decided to provide a format of file easily readable by any device (computer, mobile phone, web browser ) connected to a network. For this reason, we chose to send the shortest path description via a XML document.

As the size of the document isn’t an issue (path.xml << 500 Ko), we chose a “DOM” (Document Object Model “DocumentBuilderFactory” object to create a document designed to handle small files. The “DocumentBuilderFactory” is well suited to handle small and medium XML files (<10megs)

```
// first of all we request out DOM-implementation:
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
// then we have to create document-loader:
DocumentBuilder loader;
loader = factory.newDocumentBuilder();
// creating a new DOM-document...
document = loader.newDocument();
```

Once the shortest path XML file has been generated, the server will serialize it and send the file through the internet to the client application.

As we have now studied the design the mobile phone and of the server. We're now going to study how we can connect these two devices each other's. For this reason we will study in the following part the interaction between the server and the mobile phone application.

## 4) Interaction Server / Mobile phone

---

In this part, we are going to have a closer look at the exchanges between the application on the mobile phone and the server. It represents one of the most essential parts of the project. Indeed, without this connection, the user can only browse the map, and not have his route calculated, which is the main purpose of the program.

### 4.1) Exchange protocol

For the mobile phone to ask a route calculation, and the server to answer it, we need to send and receive information on both sides. To have a stable system, we had to set a protocol defining which side will send data at a precise moment, in which order, and deal with the error each system could face.

Here is the exchange sequence we agreed on. The example shows the information sent and received when the user asks for a route from an address to another address:

```
Client: "FROM ADDRESS TO ADDRESS"
- Server: "OK"
  Client: starting street (string) + starting city (string) + starting country (string)
  Client: arrival street (string) + arrival city (string) + arrival country (string)
    - Server: "OK"
      Server: XML path file
      Client: "OK"
    - Server: "WRONG STARTING ADDRESS"
      Client: "OK"
    - Server: "WRONG ARRIVAL ADDRESS"
      Client: "OK"
    - Server: "WRONG STARTING AND ARRIVAL ADDRESSES"
      Client: "OK"
    - Server: "PATH NOT FOUND"
      Client: "OK"
  - Server: "NOT READY"
    Client: "OK"
```

As you can see, we put error controls all along the exchange, so that if the server encounters a problem during the transfer, the application will not stay waiting endlessly, or in an infinite loop.

Technically speaking, we used sockets to link the two applications. When using on the mobile phone (and not the emulator), this connection is handled by the telecom operator via the 3G. The stability of this connection cannot be assured, and it is the main issue we had regarding this interaction.



## 4.2) XML file

The route sent by the server to the mobile phone has a lot of information: details of the whole trip, and also details on each step composing it. It would be way too complicated to send these elements one by one. That's why, after browsing different possibilities, we ended up with a few feasible options: send a text file, or a XML file. The first one would be lighter but really hard to write first, and decrypt then. Some of us having used XML in previous projects, and after investigating even more on this solution, it appeared that this solution was the most efficient.

We tried to make the structure of the XML file as simple, complete, and explicit as possible. For that we decided to split the route in blocks, and give properties to all these blocks. Here is the structure of the route:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<path bustime="774" date="2008-12-16" generator="AAU_BUS_ROUTER_SERVER" nbustaken="1"
speedwalk="1.4277777777777778" time="15:12:59" walktime="1589">
  <step mode="walk">
    <segment angle="0" distance="319.8222222222222" eta="15:16:6" etd="15:12:22"
idsegment="27927943" name="Bertil Ohlins Vej" nbWayBefTurn="1">
      <node idnode="312120262" lat="57.014772" lon="9.9842716" />
      <node idnode="312163260" lat="57.0148037" lon="9.9833801" />
      <node idnode="312493799" lat="57.0148208" lon="9.9828978" />
      <node idnode="306553286" lat="57.0148623" lon="9.9817288" />
      <node idnode="306553284" lat="57.015353" lon="9.9808276" />
      <node idnode="312493796" lat="57.0153746" lon="9.9801681" />
      <node idnode="312163572" lat="57.0154029" lon="9.9793034" />
    </segment>
  </step>
  <step mode="bus">
    <segment busref="2" direction="Lindholm Station" eta="15:29:0" etd="15:16:6"
idsegment="51471" instop="AAU Kroghstræde" outstop="Østre Allé">
      <node idnode="312163572" lat="57.0154029" lon="9.9793034" />
      <node idnode="312164357" lat="57.0153051" lon="9.9763292" />
      <node idnode="76551475" lat="57.0169443" lon="9.9724803" />
      <node idnode="63719137" lat="57.0191871" lon="9.9682574" />
      <node idnode="312235248" lat="57.0211349" lon="9.9525687" />
      <node idnode="27475686" lat="57.0247111" lon="9.9424967" />
      <node idnode="314945115" lat="57.0279366" lon="9.9443278" />
      <node idnode="27475672" lat="57.032949" lon="9.9426884" />
      <node idnode="314946349" lat="57.0380864" lon="9.9358743" />
    </segment>
  </step>
  <step mode="walk">
    <segment angle="242" distance="8.566666666666665" eta="15:29:6" etd="15:29:0"
idsegment="28423376" name="Bornholmsgade" nbWayBefTurn="1">
      <node idnode="314946349" lat="57.0380864" lon="9.9358743" />
      <node idnode="29785605" lat="57.0380162" lon="9.9359336" />
    </segment>
    <segment angle="85" distance="272.70555555555555" eta="15:32:17" etd="15:29:6"
idsegment="4525684" name="Østre Alle" nbWayBefTurn="1">
```

```

    <node idnode="29785605" lat="57.0380162" lon="9.9359336" />
    <node idnode="29785606" lat="57.0379912" lon="9.9358503" />
    <node idnode="29785607" lat="57.0378349" lon="9.9334752" />
    <node idnode="28784981" lat="57.0377879" lon="9.9316569" />
    <node idnode="28784980" lat="57.0377937" lon="9.9314441" />
  </segment>
  <segment angle="95" distance="522.5666666666666" eta="15:38:23"
    etd="15:32:17" idsegment="8041859" name="Sønderbro" nbWayBefTurn="1">
    <node idnode="28784980" lat="57.0377937" lon="9.9314441" />
    <node idnode="28784977" lat="57.0391179" lon="9.9317381" />
    <node idnode="28784976" lat="57.0395361" lon="9.9317808" />
    <node idnode="28784975" lat="57.0396624" lon="9.9317555" />
    <node idnode="28784974" lat="57.0403779" lon="9.9314609" />
    <node idnode="28784973" lat="57.041085" lon="9.9311495" />
    <node idnode="28784972" lat="57.0419856" lon="9.9307539" />
    <node idnode="28784970" lat="57.0424495" lon="9.9305612" />
  </segment>
</step>
</path>

```

#### 4.2.1) Path

It is basically the whole route. It contains general information about the trip, such as:

- "bustime": time (in seconds) spent in the bus
- "date": date at which the document has been generated
- "generator": application that has generated the file (always "AAU\_BUS\_ROUTER\_SERVER" in our case)
- "nbbustaken": number of different buses taken during the trip
- "speedwalk": speed (in meters/second) used for the calculation of the walking durations
- "time": time at which the document has been generated
- "walktime": time (in seconds) spent walking

#### 4.2.2) Step

A step split the trip in walking parts, and bus parts. Every time the user has to pass from one to the other, a new step is created. The main use is for the phone to be able to distinguish the current transportation mode and to give the information the corresponding information. That's the reason why it only has one parameter:

- “mode”: whether the user has to walk or to take the bus on that part of the route

#### **4.2.3) Segment**

This is the mode significant element of the route. It will separate each street, hence it will determine the moments when the user must be provided with information.

If we are currently walking (step mode = walk), the segment corresponds more or less to a street: it’s the way between two intersections. Here are the parameters given, then:

- “angle”: the angle between the end of the previous street and the beginning of the current one. It’s used to tell the user to turn left, right, etc.
- “distance”: distance to walk on the segment, until the next intersection.
- “eta”: stands for “Estimated Time of Arrival”; time when the user is supposed to start walking along this segment
- “etd”: stands for “Estimated Time of Departure”; time when the user is supposed to finish walking along this segment, when he reaches the next one.
- “idsegment”: identifier of the segment used by the server. In our case, the mobile phone doesn’t use it at all, but in the perspective that another system can connect to our server, it could be helpful.
- “name”: basically the name of the segment, which is, most of the time, a street.
- “nbwaybefturn”: the number of ways to cross before the next change of direction.

In the case of a bus period (step mode = bus), the segment is a “bus moment”: when the user will take a certain bus line, to a certain direction, from a bus stop to another. Within the same step (bus step), the user may take several buses in a row, each of them defined by a separated segment. The parameters hence differ from the one for a walking segment:

- “busref”: bus line the user has to take
- “direction”: direction of the bus. The one of the two ends of the line the user has to head to.
- “eta”: stands for “Estimated Time of Arrival”; time when the bus is supposed to arrive at the bus stop the user leaves at
- “etd”: stands for “Estimated Time of Departure”; time when the bus that the user takes is supposed to leave at

- "idsegment": identifier of the segment used by the server. It has the same role as in the walk mode
- "instop": name of the bus stop the user gets in the bus
- "outstop": name of the bus stop the user gets off the bus

#### **4.2.4) Node**

The node is the smallest unit of a path. Because roads are defined by a succession of segments, we need to have all the points composing the segments. So a segment will be composed by a number, small or big, of nodes. These ones are only defined by a few arguments:

- "idnode": identifier of the node. Only used by the server when calculating the route
- "lat": latitude of the point
- "lon": longitude of the point

To sum up on the XML file, we can say that the structure used allows the mobile phone to have the pertinent information on each unit of the route. It makes the treatment faster and easier for the phone. First, it is easier to receive, but the information transported is optimized as well.

Once all the solutions and choices made during the design phase had been validated, we began the implementation of the system. In the next part of the report, we will describe the development issues, solutions and design modifications made during the implementation period.

## 3

# Implementation

During the implementation phase of the project, various problems or design improvement ideas were encountered. For this reason, we will now present the design modifications that occurred while we were working on the project. We will study in a first part the implementation of the mobile phone application and in a second part the implementation of the application server.

## 1) Mobile phone

---

As we have seen in the previous parts, the project can be divided into several parts. From a technical point of view, we only developed the Mobile phone application, and the server application. Our project uses other elements, as it was explained previously, but those elements belong to other systems (like the GPS). In the implementation part, we will only explain how we developed the mobile phone and the server applications.

### 1.1) Issues met during the development

In this part, we will focus on the issues we met during the development of the mobile phone application. We will also explain how we solved those problems, and the choices we had to make to apply those resolutions.

#### 1.1.1) Emulator

- Choice of the emulator

One of the biggest issues we met during the development was the emulator. To program an application that would run on the mobile phone, we used the software Eclipse with J2ME (Java 2 Micro Edition). But in order to simulate the application running on the mobile phone, we had to use an emulator.

We found two emulators available on the internet. Both of them worked as emulators for the Nokia S60 series, which was what we were looking for. The first one was an emulator developed by Sun, and the second one was developed and used by Nokia. After some tests, we decided to work

with the Nokia emulator. Indeed, the Sun emulator did not work for some codes we developed, and it seemed less easy to use.

- Problems with the Nokia emulator

However, the Nokia emulator was far from perfect. During all our development on the mobile phone application, we faced a lot of unexplainable errors, and very long loading times. This was really an issue, since our only way to test our application was to run it on the emulator. It was an issue we did not expect when we planned the implementation of our project.

Another important problem with the differences between the emulator and the device it was supposed to emulate. We indeed noticed several functions that worked perfectly on the emulator, but did throw an error or an exception on the real device. Fortunately, we noticed this issue quite early in our development, and we thus decided to test on the real device from time to time. The mistake would have been to develop the entire application based on the tests with the emulator, only to see that it is not working on the mobile phone itself. Several time, we had to face errors on the device, despite the fact that the program was running flawlessly on the emulator. It was quite hard to understand the differences between the emulator and the device.

The last problem was the error feedback. In case of a program crash, the emulator was indeed unable to give the number of the line where the program crashed. This could be really a problem, since we always lost some time trying to find the error in the application. The only feedback the application gave was the name of the Exception or the number of the error triggered, which was quite insufficient.

### ***1.1.2) Main Menu rotation***

One of the first elements we started to implement was the main menu of the mobile phone application. It was really important for us to create a good-looking, well-designed menu, with an efficient and fast animation. As said in the Design part, we thought about a menu with four icons. Each icon would be associated to a label and an action. To select the icon corresponding to the action he wants to perform, the user should use the left and right arrows to make a rotation of the icons. That meant we needed to perform at least two different animations. The main problem was that the mobile phone did not show very good performance with graphical objects, and that is why we trying many different ways to program this menu.

- Java animation

Our first attempt was to program this menu in Java only. We inserted the four icons as images, and we used Java functions to create the rotation. Unfortunately, it was a bit slow, especially on the phone. We thus wanted to find a better way to create the animations.

- Scalable Vector Graphics (SVG)

We decided that we needed something more appropriate for this kind of small animations. After some research, we found out that Scalable Vector Graphics (SVG) technology seemed the most efficient way of programming this.

SVG is a XML-based technology, and its purpose is to describe an image with vectors. The advantage of this method is that a SVG image can be made bigger or smaller, without changing its quality. We decided that we could use SVG images in our menu. Moreover, we could also use SVG animations for the rotations.

Unfortunately, this technology did not work as well as planned on the mobile phone. We did find the SVG images on the internet, and we managed to write the animations. But the problem was to play the animation at the right time, and to choose between several animations. We tried to play only a part of the SVG file, but it was not possible. Then we tried to use the Java code to write inside the SVG file, but that was a dead end as well. That is why we decided to abandon the idea of using this technology, after spending quite a lot of time on it.

Thus, we went back to a Java-made animation. We reduced the number of frames, as well as the scaling of the icons. We also improved the efficiency of the code. The animations were running much faster, and the result was satisfactory.

### ***1.1.3) Threads***

As it is understandable from the description of the code, the mobile phone application is quite complex, in the way that it has to interact with the server and the user at the same time. This means that we had to use several threads in the program.

- The Main Thread

This thread is the main one, which means it will ensure that the user interface reacts to the user's interactions, and that it will manage most of the data processing needed in the application. It shows the right screens and let the user browse the application. It also saves the changes in the favorites and settings records. It also displays the map and the path, when the user asks for guidance.

- GPS Thread

The application is in need of GPS information for a lot of its functions. When the application needs the coordinates of the user, or some other information provided by the GPS, it will start the GPS thread. First this thread tries to find a GPS device connected to the phone. If he does, it will retrieve all the available information, and update the program variables.

- Connection Thread

This thread ensures the communication between the mobile phone and the server. It creates a socket and sends the information and the parameters needed for the path calculation to the server.

- Wait Thread

When the user asks for a path calculation, a pop-up screen appears, with a progress bar, to tell him to wait some time for the process to be finished. This screen is created and managed by the Wait thread.

- Synchronization of the threads

The hard part was actually to synchronize those different thread, at the right times, in order to get a fast application. The main thread must indeed be synchronized with the GPS and Connection thread, and the application must keep the connections with the server and the GPS all time.

#### ***1.1.4) XML Parsing***

The purpose of the mobile phone application is to give the shortest path to the user. This path is given by the server, in a XML file. One of the issues we faced was to parse this XML file, in order to retrieve the information.

First, we tried to use the parser integrated in J2ME, but we had to abandon this idea because of some problems we encountered. There were indeed some problems with the identification of the mark-ups in the XML file.

For this reason, we had to find and install another parser library. We chose KXML parser. It worked as we wished and it enabled us to exchange information via XML files between the server and the mobile phone.

## **2) Application Server**

---

As told in the design part, we developed the application server using Java and XML language. We used Eclipse IDE (integrated development environment) to implement the java classes and to edit the XML files. We can divide the implementation in four parts:

- Development of the map data parsing
- Development of the geocoding method
- Development of the shortest path method
- Development of the schedule retriever method

No big issues were encountered during the development of the Geocoding, Map parsing and shortest path methods. On the other hand, the schedule retriever implementation was more complex than expected because of the lack of information concerning the bus schedules. For this reason, we will study in the following part the solutions used to retrieve the most accurate bus schedules.

### **2.1) Schedule Retriever**

In the implementation of this block of the program, we faced some problems, mainly concerning the NT's website. We had two main problems:

- The information in NT's website is only for the next 2 hours.



- The information in each bus stop page only displays 10 buses. To see the following buses, the program has to check the next page of the bus stop schedule.
- For each bus stop, only the departure time of the bus is displayed. At some bus stops, arrival and departure times may be different.

### 2.1.1) The time limitation

In most of the cases, the user is not going to specify a departure time to calculate the path and the server will consider that he wants the best route at the query reception time. But, we also considered the case when the user wants to specify a departure time. As the information presented on NT's website is limited to the next 2 hours, we decided to keep the schedules in a XML database.

To retrieve these schedules, we developed and used a program during 24 hours to retrieve the information from NT's website. But this is not enough, since during Saturdays and Sundays the schedule changes. Thus, we implemented a program, called *BusChecker* and executed it during 72 hours (Saturday, Sunday and Monday), to get all the schedules of the busses.

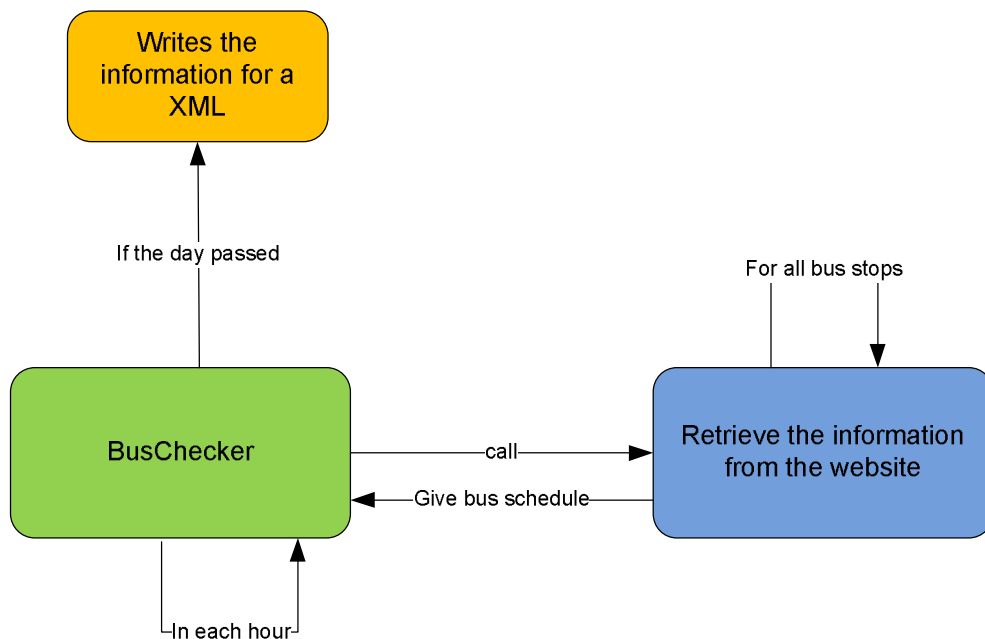


Figure 55 - *BusChecker's* flow

1 – The program was running nonstop during 72 hours but it retrieves only hour by hour the information to restrict the network traffic.

2 – After getting the information from the website, each bus schedule is only added to the list if it isn't in the list yet.

3 – After midnight, the program writes the entire list to a XML file. The information present in the XML file is organized by bus stops nodes and each bus stop node contains the schedule for a specific day.

Once the complete schedules list was downloaded, we implemented in the server all the classes necessary to run the *Bus Retriever* and put in the appendixes a short part of one of them (Saturday schedule: [Appendix 3]).

Finally, we had to decide when we had to invoke the information from the XML files or to download the data directly from NT's website. We checked the information from NT's website and we realized that there were no delays for buses departing one hour after the query time. In this case, it isn't necessary to retrieve the information from the website from this limit on. And since the access to a XML file is much faster than the access to a webpage, we decided to download the schedules from the NT website only if the difference between the arrival time at the bus stop and the current time was less than one hour.

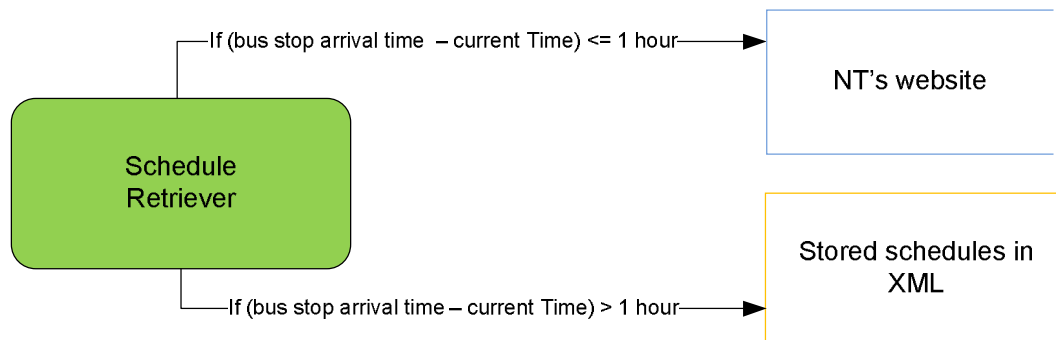


Figure 56 - Retrieving the information from different sources

### 2.1.2) The offset's problem

When the program requests the information directly from NT's website, there is another problem: a page only displays the first 10 buses. To retrieve more schedules, it is necessary to display the next page. The next page can be downloaded by adding in the URL an offset, starting at 10 (e.g. <http://ntlive.dk/rt/destination/8519915?offset=10>). For this reason, when no bus of a bus line is found in the first page, it is necessary to retrieve the schedule in the second page. If in the second page, the program still doesn't find any schedule, it is necessary to get the third one, and so on. We had to use the same method to retrieve the schedules of bus departing 30 or 40 minutes after the query time.

Thus, we modified the code retrieving the information from the webpage. Instead of simply retrieving the first page, we put the program in a loop until it finds some schedules. After that, we applied the same procedure: the first bus that arrives to the bus stop after the user arrives is the one that is returned to the shortest path calculation method.

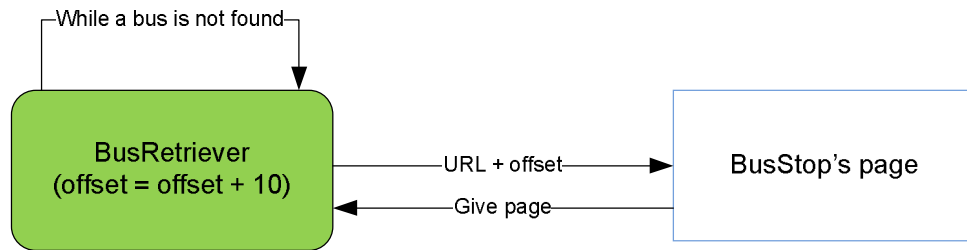


Figure 57 - The problem of the offsets

We will present, in the next section, the structure bus retriever method.

### 2.1.3) Retrieving from the next bus stop

When the bus retriever method is called by the shortest path calculation method, it receives:

- an id of the bus stop ("A") and of the bus line to test
- a time of arrival to this bus stop

It has to check that the user has enough time to get into the bus stop ("A") and to return the first arrival time at the next station ("B").

But it appeared that, using the information provided by the NT website, we couldn't give a reliable arrival time. Indeed, we know only the bus departure time at a bus stop and not the arrival time. For this reason, we consider that the arrival and the departure time at a bus stop are the same. But, this approximation can introduce some errors in the shortest path calculation.

Example:



Figure 58 Error introduced by the absence of the bus stop arrival time at a bus stop

Let's consider two bus stops A and B. The bus time departure at A is 12:01, the (real) bus time arrival at B is 12:02 and the bus time departure at B is 12:10. The user arrives at A at 12:00 and wants to go to B. The walking length between A and B is 7 minutes.

As we consider in our program that the bus arrival time at a bus station is equal to the departure time, we will consider that the arrival time at B is 12:10 (+8 minutes compared to the real time). As it takes only 7 minutes by walk to go from A to B, the program will deduce that it is faster to go to B by walk. The shortest path will be:

(A → B by walk Departure Time: 12:00 Arrival Time 12:07) instead of

(A → B by bus Departure Time: 12:01 Arrival Time 12:02) if we could use the real arrival date.

In most of the shortest path calculation cases, this problem isn't important because arrival and departure time are similar but we have to keep this issue in mind. The unique solution could be to obtain from the NT bus company an access to their bus positioning servers and retrieve the real arrival and departure times.

#### 2.1.4) Final structure

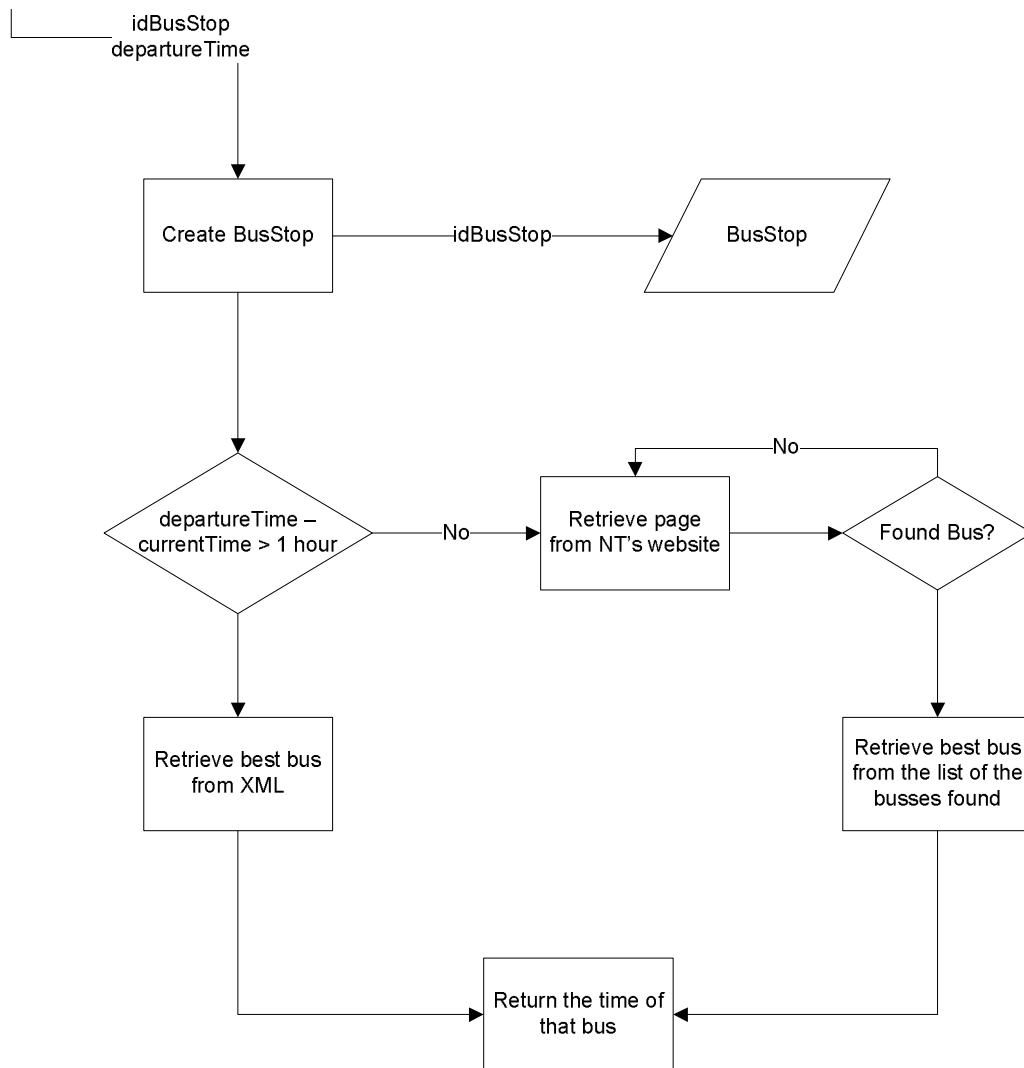


Figure 59 - Final structure of the bus schedule retriever

As we explained in the Design part and also in this part:

1 – When the *BusRetrieve* is called, it received 2 main arguments (id of a bus stop and arrival time). With the *idBusStop*, the program creates an object *BusStop*, that will get all the buses found when the information is retrieved from NT's website.

2 – As we explained before, if the difference between the arrival time received and the query time is longer than one hour, the program uses the information from a XML file. Otherwise, the program will send some queries on NT's website, until a bus with the correct bus number, destination and schedule is found.

3 – Finally, the arrival time at the next station is returned to the shortest path method.

Throughout the implementation of the system, we tested all the code blocks separately and globally to insure that the final system will be fully functional. We will study in the next part all the tests made to assure the reliability of the whole system.



## 4

# Testing

In this section, we'll talk about all the tests performed for the final application, both server and mobile application. Also, we'll refer the results obtained and the corrections made after the tests.

## 1) List of tests

---

It is true that we made tests before, for the LoFi and HiFi prototypes, but was always in a lab, and since we couldn't test the entire path just staying in the same place, we decided to perform more complex tests for the final application. In the following list, it is described the list of tests:

1 – Retrieve the path from a current position (e.g. just outside of our group room) to the city center. Look to the entire path, change between graphic and text mode, and follow the instructions given by the application.

2 – When the user arrives, retrieve the path to the University, this time with the sound activated. Follow all the instructions given by the program.

3 – Change the language of the application.

4 – Look at the map around the university.

5 – Add the Student House to the favourites (Gammel Torv, 10).

6 – Retrieve the path from the university to the Student House. Get far from the path indicated by the application, and wait for a new path.

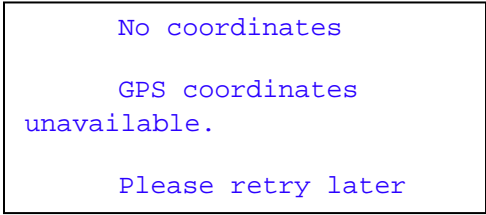
## 2) Results of the tests

---

After we completed the project, we started to perform the final tests in a morning of our work in the University. Nevertheless, we started to run the application server in the university, via connection with the Remote Desktop Connection. After that, two of the elements of our group got

out with the Nokia N95, and started the mobile application *BusRouter*. With the list of tests in mind, they follow all the steps in the list. Thus, this was what it happens in that morning of tests:

1 – After leaving out from the building (Niels Jernes Vej 12), they selected the option **Destination** in the main menu, and they filled the Destination street field with “Boulevarden”. Since the check box “Start from the current position” was already checked, they just pressed **Options > Go to**. Since in that moment the mobile was without GPS signal, they received the following message:



```
No coordinates
GPS coordinates
unavailable.
Please retry later
```

This was the first problem that we noticed in these final tests: near buildings it is hard to get a GPS signal. Since they knew the address of the building, they just unchecked the previous check box, and put “Niels Jernes Vej” as Start street. After this they did the same procedure (pressing **Options > Go to**), this time the mobile application was communicating finally with the application server. In the group room the rest of the group was viewing the data coming on the computer screen. With this data it was everything as we expected, and after all the calculation performed by the application server, the xml with the path had been sent to the mobile application.

Finally, the members of our group outside were seeing the route on the mobile screen device. They started walking to the bus stop indicated in the map. After some seconds, the mobile phone started to send the GPS signal and their position was, from that moment on, frequently updated in the route. As this first test was also to test the text mode, they pressed **Options > Text mode** and they had on the screen the path written. In the bus stop, they took the bus number 2 as indicated on the mobile’s screen.

During the bus trip, they lost sometimes the GPS signal. This problem is in many ways similar to the previous one: near or inside indoors places, the GPS signal is not so accurate or it doesn’t exist at all. Nevertheless, they got out in the correct stop, and since the final bus stop is in Boulevarden, the program, after receiving again the GPS signal, just informed on the screen that they have arrived.

2 – In this second test, it was time to test how works with the sound activated, and this time in the opposite way, i.e. to the University. They selected again **Destination** in the main menu, and this time they checked the option “Start from the current position”. After that, they just filled the Destination Street with “Niels Jernes Vej”. After this step, they pressed again **Options > Go to**, as in the previous test, and this time, instead of the information displayed on the screen, they started to listen the instructions given by the *BusRouter*.

One more time, the GPS signal has been lost during the bus trip. However, the GPS signal has been recovered and finally they listened that was necessary to get out in the next bus stop. And it



was what they did, following the rest of the instructions from the bus stop until the building where it works our entire group.

3 and 4 – Again in the group room, everybody played a little bit with the application, changing the language or viewing the map around the university. Nothing wrong happened with these functions of the mobile application.

5 – This test was made without problems again. One of us just selected **Favourites** in the main menu and after **Options > Add**. He just filled all the text fields with the entire information about the address and gave the name “SH” to this favourite. Finally, he selected **Save** and after this favourite appeared in the favourites menu.

6 – Finally, one of our group elements went out of the building to retrieve a new path to the Student House. He selected **Favourites** in the main menu, and after that, with the “SH” highlighted, he selected **Options > Go to**. After sending the information to the application server and retrieving back the path, he went in the opposite way than the indicated in the path. Since the GPS signal was working, the mobile application detected that the user was lost and sent new coordinates to the application server, that it retrieved another path to the Student House.

Now that we showed a general view of how we made the tests and also the problems faced while performing them, we’ll talk about how the project was in all of its aspects, what was like we expected and which problems we didn’t avoid or solve.



# Conclusion

## 1) Project Outcome

---

We will now look deeper into the outcome of this project, and analyze what we did and what we did not achieve, compared to our initial objectives.

First, we will list the parts of the project that we managed to develop as planned.

- The path calculation we programmed on the server works well. It is able to give a path with a good timing, using the bus schedule and the information of OpenStreetMap. The bus schedules are retrieved by the server on NT website.
- The server and mobile phone are able to exchange information and the communication protocol is designed to report any errors.
- The mobile phone application is able to display the path it receives from the server, in a clear and precise way. It is also able to guide the user to his destination, thanks to voice messages.
- The user interface of the mobile phone has been completed as planned. It allows the user to interact efficiently with the system.

The main objectives of our project are thus been completed. The whole system is working, and is able to guide a user through Aalborg city. However, there are some minor issues that could not be solved, either because of a lack of time or because of unexpected technical issues.

- We had planned to put some parameters to the shortest path calculation, and to let the user set them. For example, the user could specify that he wanted a path with the smallest walking time. We thought it would be a feature we could easily implement in the Dijkstra algorithm. However, it turned out that, due to the complexity of the shortest path calculation algorithm, it was not possible to implement this kind of parameters in the way we thought.
- We could not get in touch with NT Company until very late in the project schedule. When we finally received an e-mail, the whole system was already nearly completely implemented. However, some other information could not be obtained, like the bus stations coordinates. We had to make coordinates measures ourselves, which is why our program takes into account only a few bus lines.
- Since we had neither the bus lines coordinates of the entire North Jutland region, nor the computing power to handle the path calculation in the whole region map, we considered that our prototype would be limited to Aalborg city. This is not really an issue, rather a limitation due to the fact that it is only a prototype and not a commercial product.

It can be noticed that most of the issues mentioned above are due to some factors that we could not have controlled. Also, those issues are minor problems, which do not really interfere with the main goals of our project.

## 2) Personal Achievement

---

The project has required serious and organized work from each of us, but we also have learned much in many domains.

During the first part of the project, we mainly had to organize ourselves and find some good methods to work efficiently together. Most of us had already worked in team project before, but a long and important project like this one was still a good way to practice teamwork and project-oriented work. Moreover, the difficulty of the project allowed us to build a system with a complex structure. There are indeed several elements that are connected to each other, which required to write protocols of communication and to ensure that the connection is kept during the exchange of data.

Many technologies were used in the project, and each of us worked on several parts. This means that we had the opportunity to learn a lot, in many different fields. Even if our applications ran mainly on Java J2ME, we had to focus on many other technologies, like XML or flash (for the Hifi prototype). Also, we used both Google Maps API and OpenStreetMap, which are quite different, but which we used a lot and in many ways in our project. It was also very interesting to study the GPS localization.

The project was a good way to practice what we have learned in some of our courses in Aalborg Universitet. For example, we used many tips gathered in the HCI (Human Computer Interface) courses, in order to design our LoFi and HiFi prototypes, as well as our final user interface.

## 3) Perspectives

---

The project is fully working, but it does not mean that there is nothing else to do with it. Actually, we really would like it to be continued by another group. But more precisely, there are some features or elements that it would be interesting to add in our project.

- As said previously, the range of our path calculator is limited to Aalborg and its close surrounding. However, the only reason we did not fix this limit to North Jutland is that we lacked the computing power and some information to do it. With a more powerful server, and the coordinates of all the bus and train lines, it would be quite easy to make the system work on a larger area.
- We got in touch with NT at the end of the project, and they seemed very enthusiastic about it. That is why we think that some agreement could be made with NT, to get the real-time position of the busses of all the lines for example. This could improve the precision of the path calculation algorithm.
- Finally, a speech recognition module could be implemented. It would allow the user to give his destination address by telling it rather than typing it.



# References

- 
- [i] ZDNet, 2008: "Symbian vs Windows Mobile"  
<http://www.zdnet.com.au/insight/communications/soa/Enterprise-OS-wars-Symbian-v-Windows-Mobile/0,139023754,339287370,00.htm>  
Visited the 19/11/2008
- [ii] PDAsNews, 2007: "Symbian worldwide smartphone share"  
<http://www.pdasnews.com/articles/2551/1/>  
Visited the 19/11/2008
- [iii] LetsGoMobile, 2008: "Symbian vs Windows Mobile"  
<http://www.letsgomobile.org/en/review/0037/symbian-windows-mobile>  
Visited the 19/11/2008
- [iv] TNS-Sofres, 2008: "Perspectives de développement du marché des services sur mobile en France et en Europe"  
[http://www.tns-sofres.com/presse\\_communique.php?id=586](http://www.tns-sofres.com/presse_communique.php?id=586)  
Visited the 19/11/2008
- [v] Eric Vialle, 2008: "Online course –Information & Communication University GSM Localization"  
<http://blog.vialle.org/2008/05/20/geolocalisation-mobile-computing/>  
Visited the 19/11/2008
- [vi] Kowoma, 2005: "GPS-Explained"  
<http://www.kowoma.de/en/gps/index.htm>  
Visited the: 19/11/2008
- [vii] Deblauwe, N.; Treu, G., 2008: "Hybrid GPS and GSM localization energy-efficient detection of spatial triggers, Positioning, Navigation and Communication" - 2008 Volume - Issued 27 March 2008  
Page(s):181 - 189
- [viii] Scott Pace, Gerald P. Frost, Irving Lachow, David R. Frelinger, Donna Fossum, Don Wassem, Monica M. Pinto, 1994: "The Global Positioning System assessing National Policies"  
Appendix A:GPS Technologies and Alternatives - Appendix B:GPS History, Chronology, and Budgets
- [ix] Openwave, 2002: "Overview of Location Technologies"  
[http://developer.openwave.com/omtdtdocs/location\\_studio\\_sdk/pdf/Intro\\_to\\_Location\\_Technologies.pdf](http://developer.openwave.com/omtdtdocs/location_studio_sdk/pdf/Intro_to_Location_Technologies.pdf)  
Visited the: 19/11/2008

- [x] Shu Wang, Jungwon Min, Byung K. Yi, 2008: "Location Based Service for Mobiles: Technologies and Standards"  
<http://to.swang.googlepages.com/ICC2008LBSforMobilesimplifiedR2.pdf>  
Visited the: 19/11/2008
- [xi] Steve Coast, 2004: Open Street Map  
<http://www.openstreetmap.org/>  
Visited the: 19/11/2008
- [xii] Liang Dai, Carleton University School of Computer Science, 2005: "Fast Shortest Path Algorithm for Road Network and Implementation"  
<http://www.scs.carleton.ca/~maheshwa/Honor-Project/Fall05-ShortestPaths.pdf>  
Visited the: 19/11/2008
- [xiii] Thomas Willhalm, 2005: "Speed up shortest-path computations"  
<http://www.mpi-inf.mpg.de/~sanders/courses/algen04/willhalm.pdf>  
Visited the: 19/11/2008
- [xiv] Nordjyllands trafikselskab, 2008: "Nordjylland trafikselskab"  
<http://www.nordjyllandstrafikselskab.dk>  
Visited the: 19/11/2008
- [xv] NT Live, 2008: "NT Live"  
<http://ntlive.dk/rt/index>  
Visited the: 19/11/2008
- [xvi] Wikipedia, 2008: Keywords: "GPS", "Mobile Phone", "GNSS", "GSM localization", "Symbian OS"  
[http://en.wikipedia.org/wiki/Main\\_Page](http://en.wikipedia.org/wiki/Main_Page)  
Visited the 19/11/2008
- [xvii] James Alexander, 2004: "Loxodromes: A Rhumb Way to Go"  
Vol. 77 - No 5- *Mathematic Magazine* December 2004



# Appendixes

## 1) Questionnaire of the LoFi and HiFi prototype tests

---

Here are some questions regarding the test you have just performed. Please, answer each question by giving a mark and possibly writing a comment.

I)

- Gender:  
☐ M ☐ F
- Age:  
☐ 14-20 ☐ 20-25 ☐ 25-30 ☐ 30-40 ☐ 40-60
- Are you? (Multiple choices possible)  
☐ A bus user  
☐ Used to walk to go where you want to  
☐ A bike rider  
☐ A car driver

II) Questions

1) The application is easy to use:

1	2	3	4	5
---	---	---	---	---

Comments:

2) The design is user-friendly:

1	2	3	4	5
---	---	---	---	---

Comments:

3) All the features are pertinent and useful:

1	2	3	4	5
---	---	---	---	---

Comments:

4) The interfaces are logically organized:

1	2	3	4	5
---	---	---	---	---

Comments:

5) You would have liked to have more information:

Yes	No
-----	----

Comments:

6) You would use such an application on your mobile:

Yes	No
-----	----

Comments:

.....



## 2) Data.xml

---

```
<?xml version="1.0" encoding="utf-8" ?>

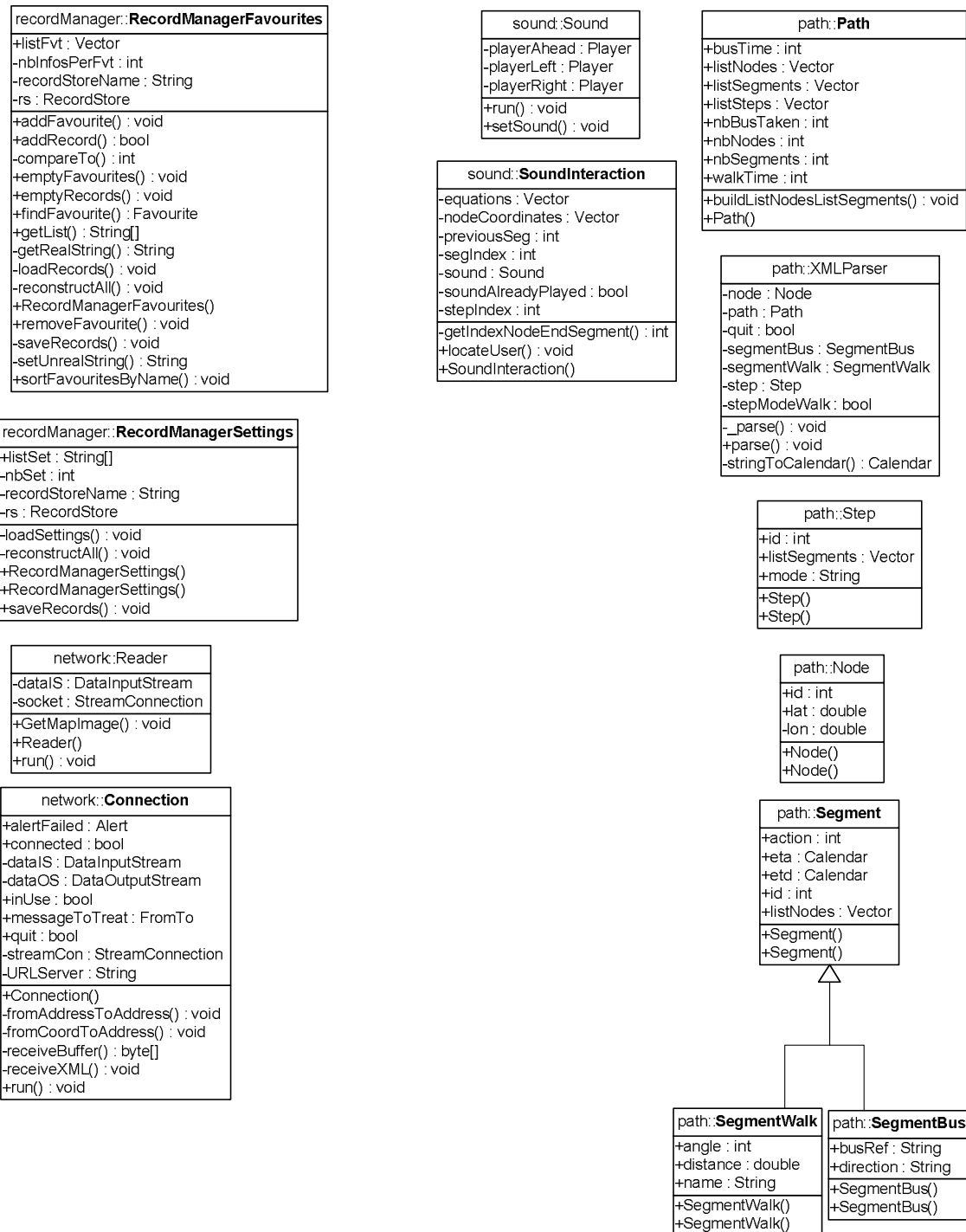
<list>
  <busStop relationId="51040" givenId="51040000" siteId="8510024" />
  <busStop relationId="51040" givenId="51040001" siteId="8518012" />
  <busStop relationId="51040" givenId="51040002" siteId="8510074" />
  <busStop relationId="51040" givenId="51040003" siteId="8510027" />
  <busStop relationId="51040" givenId="51040004" siteId="8510122" />
  <busStop relationId="51040" givenId="51040005" siteId="8512104" />
  <busStop relationId="51040" givenId="51040006" siteId="8510089" />
  <busStop relationId="51040" givenId="51040007" siteId="8511082" />
  <busStop relationId="51040" givenId="51040008" siteId="8510092" />
  <busStop relationId="51040" givenId="51040009" siteId="8510176" />
  <busStop relationId="51040" givenId="51040010" siteId="8510093" />
  <busStop relationId="51040" givenId="51040011" siteId="8510079" />
  <busStop relationId="51040" givenId="51040012" siteId="8519912" />
  <busStop relationId="51040" givenId="51040013" siteId="8519915" />
  <busStop relationId="51040" givenId="51040014" siteId="8510080" />
  <busStop relationId="51040" givenId="51040015" siteId="8519916" />
  <busStop relationId="51040" givenId="51040016" siteId="8519920" />
  <busStop relationId="51040" givenId="51040017" siteId="8510003" />

  <busStop relationId="51471" givenId="51471000" siteId="8510003" />
  <busStop relationId="51471" givenId="51471001" siteId="8519920" />
  <busStop relationId="51471" givenId="51471002" siteId="8519916" />
  <busStop relationId="51471" givenId="51471003" siteId="8510080" />
  <busStop relationId="51471" givenId="51471004" siteId="8519915" />
  <busStop relationId="51471" givenId="51471005" siteId="8519912" />
  <busStop relationId="51471" givenId="51471006" siteId="8510079" />
  <busStop relationId="51471" givenId="51471007" siteId="8510093" />
  <busStop relationId="51471" givenId="51471008" siteId="8510176" />
  <busStop relationId="51471" givenId="51471009" siteId="8510092" />
  <busStop relationId="51471" givenId="51471010" siteId="8511082" />
  <busStop relationId="51471" givenId="51471011" siteId="8510089" />
  <busStop relationId="51471" givenId="51471012" siteId="8512104" />
  <busStop relationId="51471" givenId="51471013" siteId="8510122" />
  <busStop relationId="51471" givenId="51471014" siteId="8510027" />
  <busStop relationId="51471" givenId="51471015" siteId="8510074" />
  <busStop relationId="51471" givenId="51471016" siteId="8518012" />
  <busStop relationId="51471" givenId="51471017" siteId="8510024" />
</list>
```

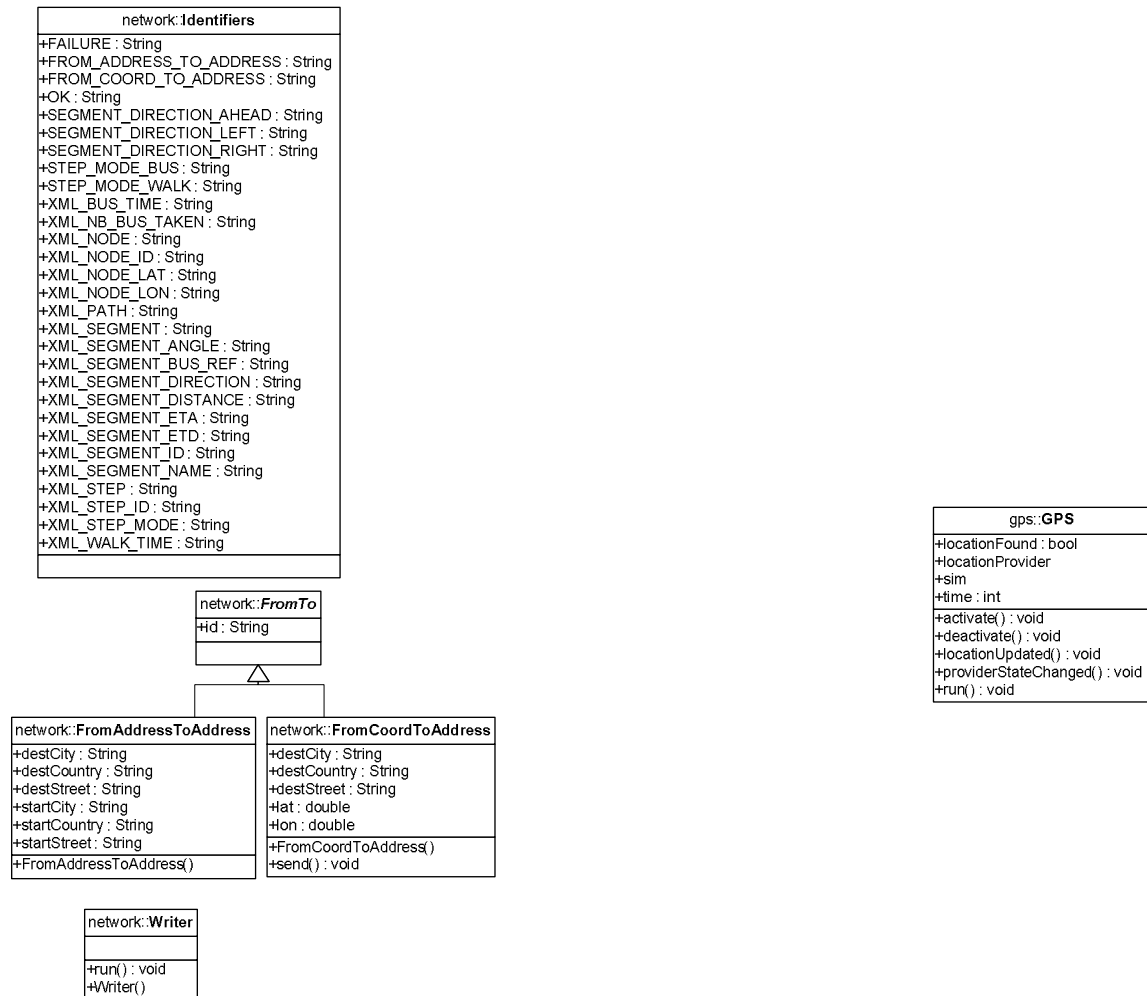
### 3) Short section of outSat.xml

[illegible]

#### 4) Class diagram of the mobile phone and server application



Mobile phone (1/5)



Mobile phone (2/5)

<b>simu::Simulation</b>
+index : int
+nodeCoordinates : Vector
+seg : Segment
+Simulation()
+updateUserCoord() : void

<b>menu::Clip</b>
-angle : int
-cmdExit : Command
-cmdSelect : Command
-coeffEllipse : float
-imgBackground
-imdDest
-imgFav
-imgMap
-imgSet
-main : Main
-offset : int
-radius : int
-x : int
-y : int
+Clip()
+commandAction() : void
#keyPressed() : void
#paint() : void
+refresh() : void

<b>menu::Menu</b>
-btnDest
-btnFav
-btnMap
-btnSet
-cmdDest : Command
-cmdFav : Command
-cmdMap : Command
-cmdQuit : Command
-cmdSet : Command
+form : Form
#ICL
-main : Main
+commandAction() : void
+Menu()
+refresh() : void

<b>applicationGlobal::Language</b>
+ADDR_CITY : int
+ADDR_COUNTRY : int
+ADDR_STREET : int
+BACK : int
+CANCEL : int
+DEST_ADD_FAV : int
+DEST_ADDR_CITY : int
+DEST_ADDR_COUNTRY : int
+DEST_ADDR_STREET : int
+DEST_GOTO : int
+DEST_MAP_MODE : int
+DEST_TEXT_AHEAD : int
+DEST_TEXT_LEFT : int
+DEST_TEXT_MODE : int
+DEST_TEXT_ON : int
+DEST_TEXT_RIGHT : int
+DEST_TEXT_TAKE_BUS : int
+DEST_TEXT_TO : int
+DEST_WAIT_TITLE : int
+DEST_WAIT_TEXT : int
+DESTINATION : int
+ERROR_CONNECTION : int
+ERROR_GOOGLE_MAPS : int
+ERROR_TITLE : int
+FAV_ADD : int
+FAV_BACK : int
+FAV_CONFIRMATION : int
+FAV_DELETE : int
+FAV_EDIT : int
+FAV_EXISTING_LABEL : int
+FAV_GOTO : int
+FAV_INVALID_LABEL : int
+FAV_LABEL : int
+FAV_NEW : int
+FAV_NO_LABEL : int
+FAV_SAVE : int
+FAV_SURE : int
+FAVOURITES : int
+MAP : int
+MAP_GPS_FOUND : int
+MAP_GPS_FOUND_NOT : int
+MAP_TITLE : int
+MENU : int
+NEXT : int
+NO : int
+OK : int
+QUIT : int
+SAVE : int
+SET_DANISH : int
+SET_FASTEST : int
+SET_FRENCH : int
+SET_LANG_DK : int
+SET_LANG_EN : int
+SET_LANG_FR : int
+SET_LANGUAGE : int
+SET_LESS_CONNECT : int
+SET_LESS_WALK : int
+SET_PATH : int
+SET_PATH_FASTEST : int
+SET_PATH_LESS_CONNECT : int
+SET_PATH_LESS_WALK : int
+SET_SOUND : int
+SET_SOUND_OFF : int
+SET_SOUND_ON : int
+SET_PATH_LESS_CONNECT : int
+SET_PATH_LESS_WALK : int
+SET_SOUND : int
+SET_SOUND_OFF : int
+SET_SOUND_ON : int
+SET_SOUND_SOUNDON : int
+SET_SOUND_SOUNDON : int
+SETTINGS : int
+STARTING_ADDR_CITY : int
+STARTING_ADDR_COUNTRY : int
+STARTING_POSITION : int
+YES : int
-strings : Language
-supportedLocales : TypeDonnées1String[]
+getString() : string

Mobile phone (3/5)

settings::Settings
<del>cmdBack</del> : Command <del>cmdSave</del> : Command +form : Form <del>language</del> : ChoiceGroup <del>pathParam</del> : ChoiceGroup <del>soundOnOff</del> : ChoiceGroup
+commandAction() : void +refresh() : void +Settings()

gps::GPS
+locationFound : bool +locationProvider +sim +time : int
+activate() : void +deactivate() : void +locationUpdated() : void +providerStateChanged() : void +run() : void

applicationGlobal::Main
<del>audioInformations</del> <del>connection</del> <del>display</del> <del>gps</del> <del>messages</del> <del>recordManagerFvt</del> <del>recordManagerSet</del> <del>screenAddrDest</del> <del>screenEditFvt</del> <del>screenListFvt</del> <del>screenManager</del> <del>screenMap</del> <del>screenMenu</del> <del>screenNewFvt</del> <del>screenSegDest</del> <del>screenSettings</del> <del>screenTextDest</del> <del>sound</del>
+commandAction() : void +destroyApp() : void +exitMIDlet() : void +pauseApp() : void +startApp() : void

applicationGlobal::ScreensManager
+list : Vector +SCREEN_DEST_ADDR : int +SCREEN_DEST_MAP : int +SCREEN_DEST_SEG : int +SCREEN_DEST_TEXT : int +SCREEN_DEST_WAIT : int +SCREEN_FVT_DELETE : int +SCREEN_FVT_EDIT : int +SCREEN_FVT_LIST : int +SCREEN_FVT_NEW : int +SCREEN_MAP : int +SCREEN_MENU : int +SCREEN_PREVIOUS : int +SCREEN_SETTINGS : int
+resetList() : void +ScreensManager() : void +showPrevious() : void +showScreen() : void +showTmp() : void +unstackLast() : void

map::Map
+alertFailed +buffer <del>cmdBack</del> : Command <del>cmdBackNo</del> : Command <del>cmdBackYes</del> : Command <del>cmdTextMode</del> : Command #coefPixPerDegLat : double #coefPixPerDegLon : double #color : string +connected : bool <del>displayPath</del> : bool #format : string #h : int -image #key : string +mapCoordinates #origin +path +pathReceived : bool #pixPerDegLat : double #pixPerDegLon : double +previousDisplayPath : bool #step : double +userCoordinates #w : int #zoom : int
+centerScreenOnPath() : void +commandAction() : void +computeURL() : void +drawThickLine() : void +keyPressed() : void +Map() +paint() : void +refresh() : void +refreshCanvas() : void

Mobile phone (4/5)

<b>destination::SegmentDestination</b>
-cmdBack : Command
-cmdNext : Command
-currentSegment : int
-distance : StringItem
-eta : StringItem
-etd : StringItem
+form : Form
-image : ImageItem
-instruction : StringItem
-calendarToString() : String
+commandAction() : void
-distanceToString() : String
+load() : void
+refresh() : void
+SegmentDestination()

<b>destination::WaitDestination</b>
-addressDestination : AddressDestination
+alert : Alert
+refresh() : void
+run() : void
+WaitDestination()

<b>destination::TextDestination</b>
-cmdBack : Command
-cmdBackNo : Command
-cmdBackYes : Command
-cmdMapNode : Command
-cmdSegDetails : Command
-iconBus : Image
-iconLeft : Image
-iconRight : Image
-iconTop : Image
+list : List
+listImages : Image[]
+listTexts : String[]
+buildLists() : void
+commandAction() : void
+refresh() : void
+TextDestination()

<b>destination::AddressDestination</b>
+actualPosition : ChoiceGroup
+alert : Alert
-cmdAddFav : Command
-cmdBack : Command
-cmdGoTo : Command
+form : Form
+path : Path
+screenWaitDest : WaitDestination
+txtFdDestCity : TextField
+txtFdDestCountry : TextField
+txtFdDestStreet : TextField
+txtFdStartCity : TextField
+txtFdStartCountry : TextField
+txtFdStartStreet : TextField
+AddressDestination()
+commandAction() : void
+itemStateChanged() : void
+loadFromFavourite() : void
+refresh() : void

<b>favourites::ListFavourites</b>
-cmdAddFvt : Command
-cmdBack : Command
-cmdDeleteFvt : Command
-cmdEditFvt : Command
-cmdGoTo : Command
+list : List
+screenDeleteFvt : DeleteFavourite
+commandAction() : void
+getFavouriteIndex() : int
+ListFavourites()
+refresh() : void
+selectFavourites() : void

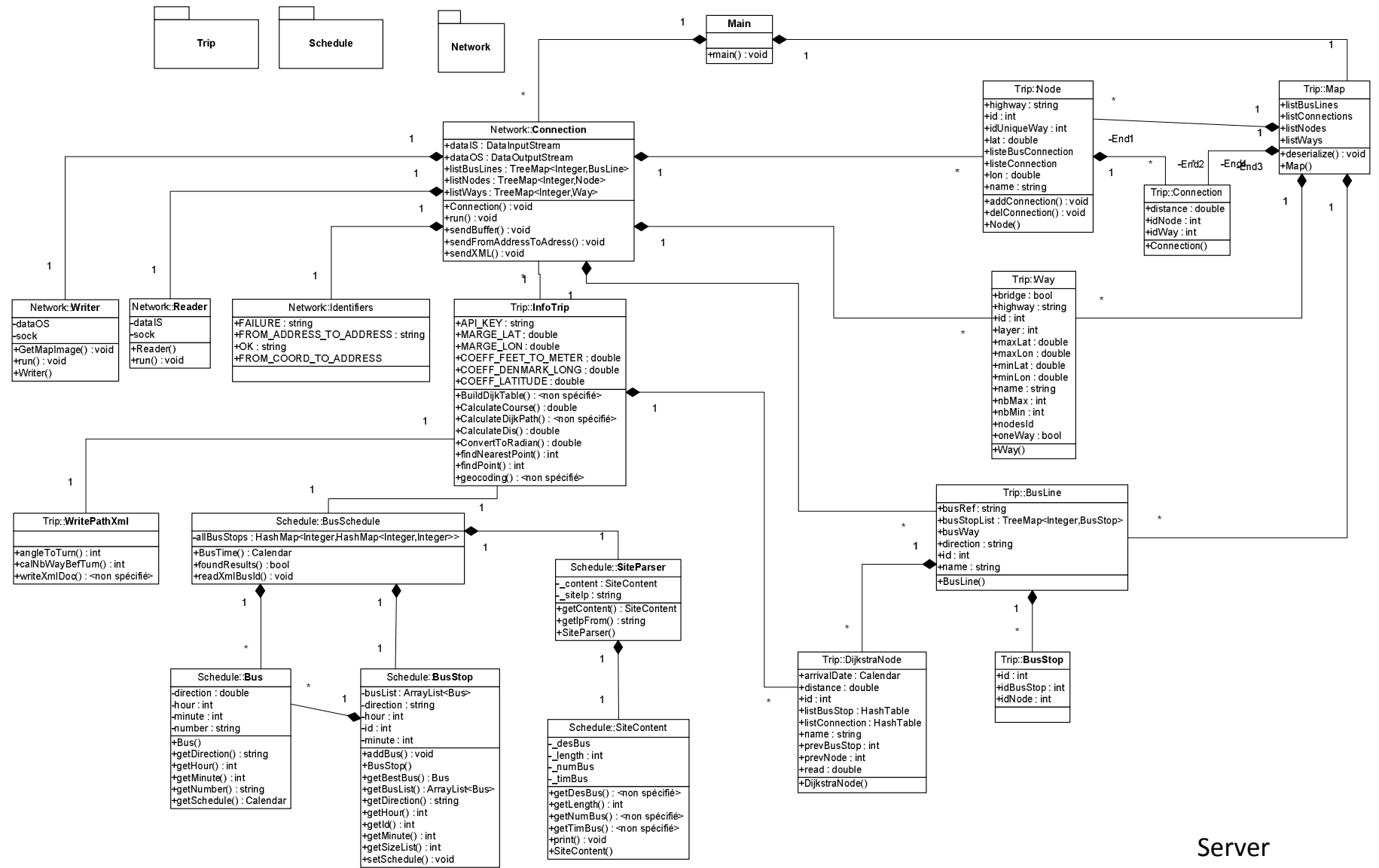
<b>favourites::DeleteFavourite</b>
+alert
-cmdDeleteNoFvt
-cmdDeleteYesFvt
+commandAction() : void
+DeleteFavourite() : void
+refresh() : void

<b>favourites::Favourite</b>
+city : string
+country : string
+label : string
+street : string
+Favourite()

<b>favourites::NewFavourite</b>
-cmdBack : Command
-cmdSaveFvt : Command
+form : Form
-txtFdCity : TextField
-txtFdCountry : TextField
-txtFdLabel : TextField
-txtFdStreet : TextField
+commandAction() : void
+empty() : void
+loadFromDestination() : void
+NewFavourite()
+refresh() : void

<b>favourites::EditFavourite</b>
-cmdBack : Command
-cmdSaveFvt : Command
-current : string
+form : Form
-txtFdCity : TextField
-txtFdCountry : TextField
-txtFdLabel : TextField
-txtFdStreet : TextField
+commandAction() : void
+EditFavourite()
+load() : ushort
+refresh() : void

## Mobile phone (5/5)



Server



