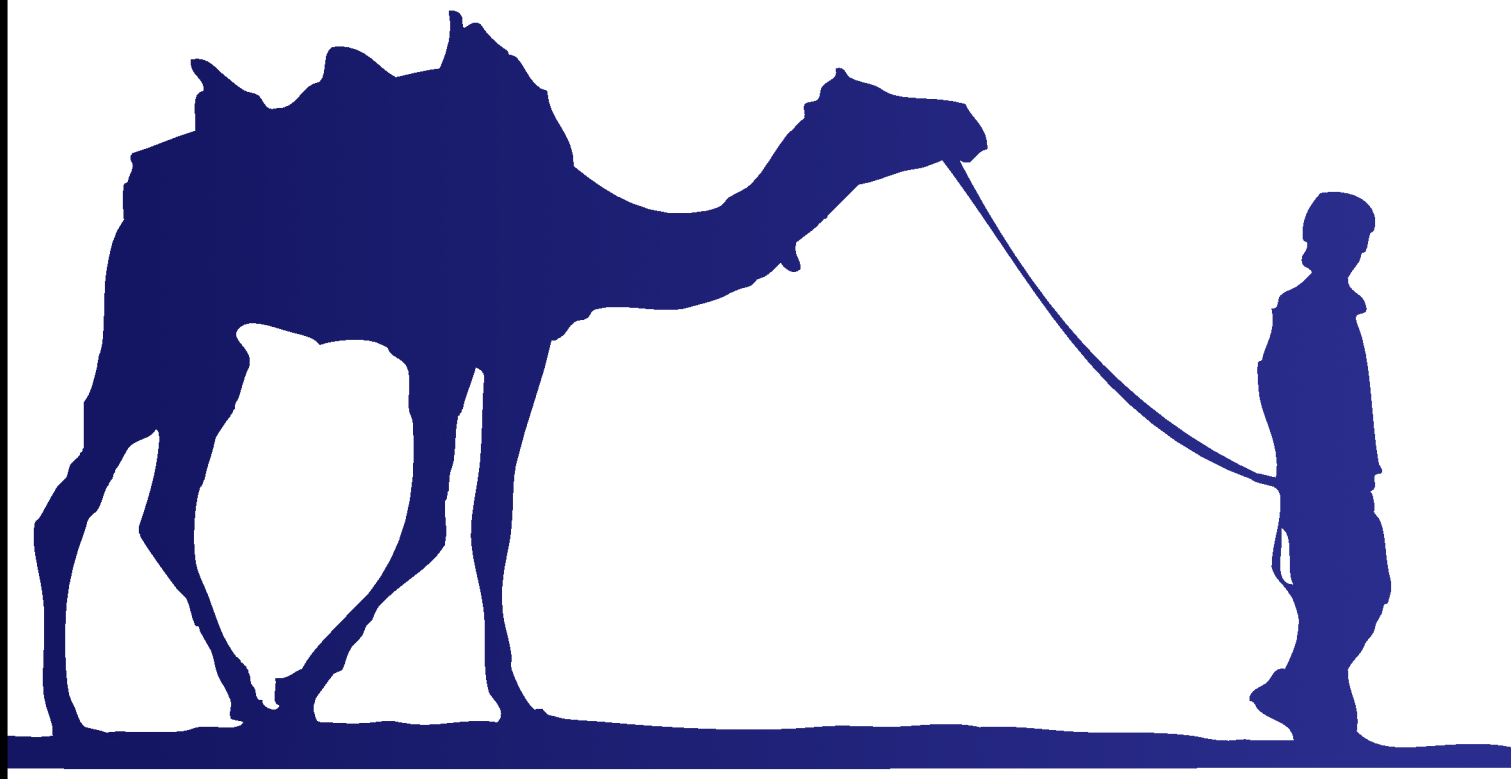
BlueCAML

Bluetooth,

Collaborative, And Maintainable

Location system



Software 8 - 2010

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg
Telephone:(45)96358080
http://www.cs.aau.dk

Title:
BlueCAML
Bluetooth, Collaborative, And Maintainable Location system

Theme:
Distributed and Mobile Software

Project time frame:
SW8, 2^{nd} Februrary - 28^{th} May, 2010

Project group:
sw801b

Group members:
Christian Frost
Casper Svenning Jensen
Kasper Søe Luckow

Supervisor:
Bent Thomsen

Abstract:
BlueCAML is an indoor positioning system targeting mobile devices. It provides the users with a map showing their current position. Bluetooth with the fingerprinting technique enables in this regard that position estimation can be performed.

Besides developing BlueCAML, a great effort is put in investigating the inherent deficiencies of the fingerprinting technique and in evaluating Bluetooth as a positioning technology. In regards to the fingerprinting technique, resolutions are implemented for addressing these. Most notably, a means is established for accommodating that environmental factors change over time. This requires maintenance of the collected environmental information, which can be a time-consuming task. Thus, an additional feature of BlueCAML is its ability to allow the users to help maintain this information.

A hypothesis is set for accommodating varying people density in the area. This involves measuring current environmental variation and remedy position estimation accordingly. However, a model describing this relationship is not set, but a foundation for further research is established.

Finally, it is concluded that Bluetooth can potentially be used as an indoor positioning system, since the achievable accuracy of BlueCAML is ten meters with a precision of 81%.

Copies: 6

Total pages: 148

Of this Appendices: 7

Paper finished: 28^{th} of May 2010

Preface

This report is written by three software-engineering students attending the 8^{th} semester at Aalborg University as a part of their semester project. The project was commenced on the 1^{st} of February 2010, and finished on May 28^{th} 2010.

During the project of BlueCAML, help has been received from a number of people who the authors would like to thank. First of all, the project's supervisor Bent Thomsen, who has been helpful with advice regarding the project and the content of the report. René Hansen gave advice, based on his experience, on the fingerprinting technique and indoor positioning in general. Laurynas Šikšnys helped in providing a reusable component for the map representation. Ulrik Mathias Nyman helped with verification issues. Kim Guldstrand Larsen helped in seeing different perspectives of BlueCAML. Jørgen Back Andersen and Rasmus Krigslund helped in understanding how radio signals behave indoors. Finally, Søren Juul and Morten Bested functioned as test subjects in the usability test.

This report will concentrate on subjects related to computer science. Therefore, it is assumed that the reader has equivalent knowledge in the field of computer science, as that of a 8^{th} semester software engineering student.

Two types of source references are used throughout the report. One is a reference placed after a period which refers to the given section. The other type of reference is placed before a period which refers to the particular sentence or word. The sources of the references used throughout this report, can be found in the bibliography at the end of the report.

Aalborg, May 2010
- sw801b

Christian Frost

Casper Svenning Jensen

Kasper Søe Luckow

Contents

1 Introduction

The applicability of outdoor positioning systems, such as the Global Positioning System (GPS), has given rise to a wide range of applications and products. These include, among others, navigation devices used in cars, watches tracking the traversed route of a person, and therefore affect many people around the world. However, GPS cannot be used for indoor positioning due to its inherent issue with radio waves being incapable of sufficiently penetrating roofs (Malik, 2009).

Based on the popularity of outdoor positioning systems, indoor positioning systems are expected to hold great potential as well, and it is therefore an area where effort is put into exploring possible solutions for such systems. Indoor positioning opens for using the indoor context of people in terms of their position to provide information accordingly. Systems build upon this concept are called context-aware and opens for numerous applications such as providing users with today's menu if they enter the canteen area (Ducatel et al., 2001).

Previously, research has primarily emphasised on exploring the potential of using Wi-Fi as the technology for indoor positioning systems (Honkavirta et al., 2009; Bahl and Padmanabhan, 2000; Hansen and Thomsen, 2009). The advantage of this is that many public buildings and institutions today are already equipped with Wi-Fi access points, hence making the deployment cost low for these. However, given that the positioning system targets mobile devices, these must as well be equipped with Wi-Fi capabilities. It is assumed that Wi-Fi support is not yet a standard feature in mobile devices. Instead, Bluetooth has been used for several years in mobile devices, and is a relatively common feature in the majority of mobile devices. Also, Bluetooth aiming at being a low-cost, energy-efficient wireless communication technology enhances the incentive of using this technology. Therefore, it is assumed that an indoor positioning system using Bluetooth would hold greater potential than systems using Wi-Fi for indoor positioning. Furthermore, previous research made by the authors (Sw701b, 2009) showed that the hardware needed for a Bluetooth infrastructure is relatively inexpensive, hence keeping the deployment cost of such an infrastructure relatively low. Therefore, this report presents an indoor positioning system called *Bluetooth, Collaborative, And Maintainable Location system* (BlueCAML), which uses Bluetooth as its positioning technology.

A problem which can be observed with some positioning techniques is a dependency on information, which changes over time, regarding the environment in which it is used. Thus,

one goal of BlueCAML is to develop, not only a positioning system using Bluetooth, but also mechanisms for enabling the users to maintain this environmental information.

In the following section, the scenario in which BlueCAML is used is described in further detail. Furthermore, an overview of related work and learning goals of the project are described. Finally, an overview of the report structure is provided.

1.1 Scenario

The purpose of this section is to describe the scenario which will form the outline of the development of BlueCAML. To formulate and understand the usage and structure of BlueCAML, the application domain work product is used to describe the work process, and the problem domain work product describes the objects in BlueCAML (Munk-Madsen et al., 2000). These help in understanding both the context in which BlueCAML is used and introduce basic concepts and entities in the system.

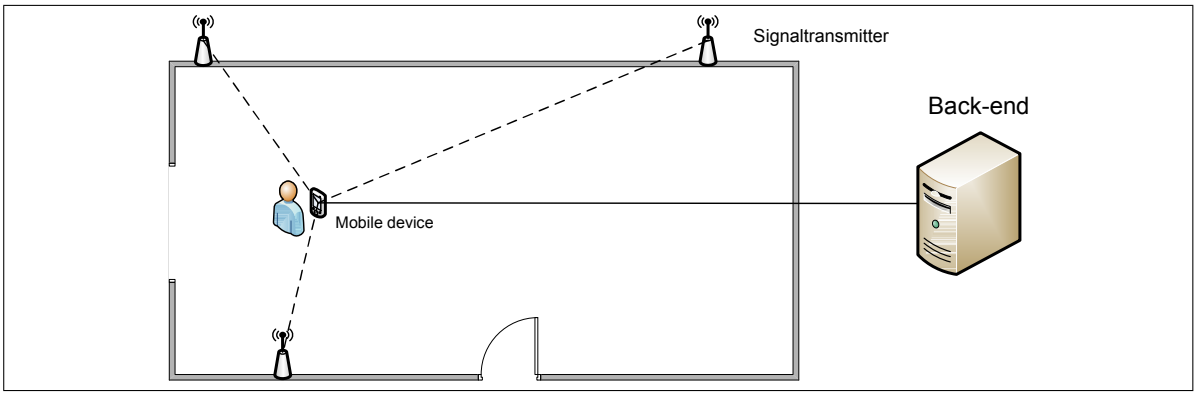


Figure 1.1: The scenario in which BlueCAML is deployed.

Figure 1.1 shows the scenario in which BlueCAML is used. The scenario consists of a number of signaltransmitters placed strategically around in the building. Users carrying mobile devices should then be able to determine their position based on a positioning technique which builds upon some relationship between the mobile devices and signaltransmitters. Furthermore, the mobile device communicates with the back-end to maintain the environmental information. Both the mobile device and signaltransmitters support Bluetooth.

1.1.1 Application Domain

The following describes how BlueCAML is used in practice.

A user enters the building with BlueCAML pre-installed on her mobile device and the building information loaded. The user is then provided a floor plan of the building with her current position which changes as she moves around in the building.

While moving around, the user may observe that inconsistencies are present on the map with respect to her actual position. In this case, the user is provided the option of correcting, and thus help maintaining the accuracy of BlueCAML. This option includes specifying which part of the map needs to be corrected and the user is afterwards given instructions on how to carry out this maintenance procedure.

Furthermore, the user is capable of requesting updated building information from the back-end which contains the corrections provided by the user herself and other users.

1.1.2 Problem Domain

BlueCAML consists of a variety of general components. The purpose of this section is to describe these, how they are interconnected, and what their purpose serve. In Figure 1.2, the components and their relationships are shown in a UML class diagram.

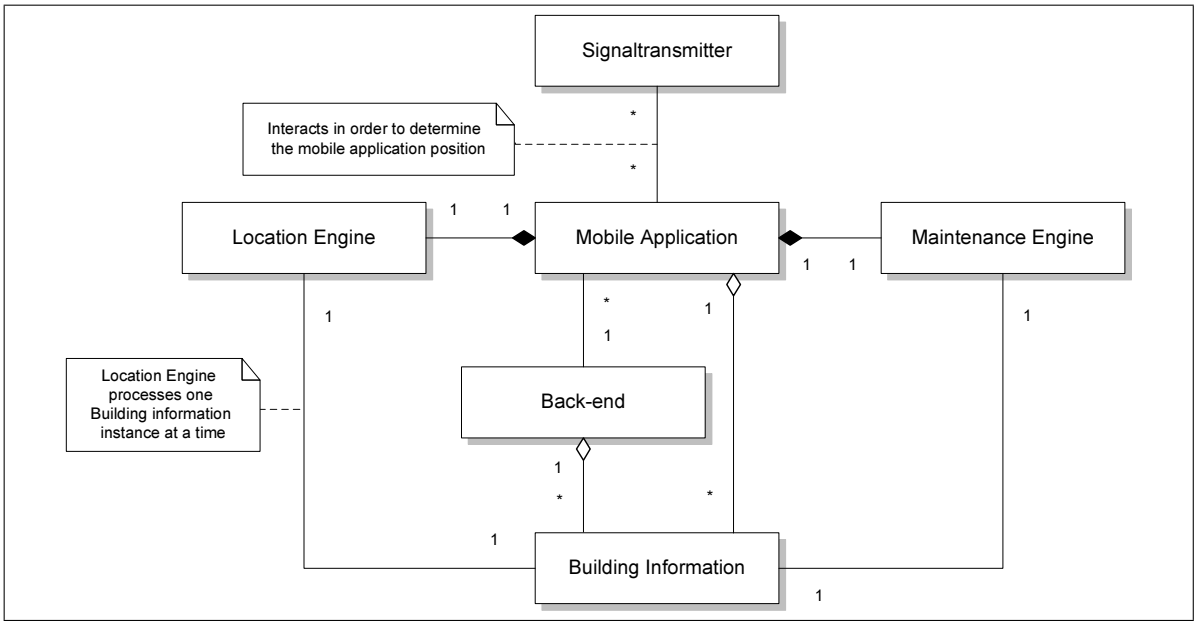


Figure 1.2: The problem domain represented as a UML class diagram.

The system comprises three major parts: the mobile application, the back-end and the signaltransmitters. The mobile application is further divided into the location engine and maintenance engine. These components, and the building information shared between multiple components are described below.

Mobile Application The mobile application is responsible for showing the map and current position to the user. It is related to one or more signaltransmitters in order to estimate its position and further consists of a Location Engine and a Maintenance Engine.

Location Engine The location engine is responsible for conducting the actual position estimation procedure of the user.

Maintenance Engine The maintenance engine is responsible for letting the user upload corrected environmental information to the back-end.

Back-end The back-end is responsible for storing the environmental information about the buildings and provides a means for users of the system to upload corrections to the building information.

Signaltransmitter The signaltransmitters are Bluetooth enabled devices which are an essential part of the positioning system. In general they provide a means of estimating the position. How these are used depends on the chosen positioning method described in Chapter 5.

Building Information The building information consists of information needed by the location engine in order to estimate positions. The actual type of building information is likewise dependent on the chosen positioning method.

1.2 Related Work

Examining existing research related to ones project is a good idea since it opens for the possibility of drawing inspiration or to reuse components.

To the authors knowledge, no systems are similar to BlueCAML, but some incorporates ideas which BlueCAML builds upon. Systems from which inspiration can be drawn are: RADAR (Bahl and Padmanabhan, 2000), Weighted Graphs (Hansen and Thomsen, 2009), BLIP (BLIP, 2010), and Easy Clocking (Sw701b, 2009) that all are described below.

1.2.1 RADAR

The RADAR (Bahl and Padmanabhan, 2000) project introduced the concept of positioning with an existing Wi-Fi infrastructure. The observations and methods used in this research have been extensively cited and build upon in further research in the field. It describes methods such as fingerprinting and propagation models to track a Wi-Fi enabled laptop in an office environment. Many of the observations and techniques it describes constitute the basis for positioning in BlueCAML combined with new techniques in accordance to advances in the field since the release of the paper on the RADAR project.

The RADAR project achieved an accuracy of 2.94 meters with a precision of 50% using fingerprinting and the Nearest Neighbour searching strategy, which are described in Chapter 5. Additionally, an accuracy of 4.3 meters with 50% precision using measured Received Signal Strength Indicator (RSSI) values and propagation models was achieved. They also researched the impact of user orientation which decreased the accuracy to 4.90 meters for 50% precision using fingerprinting.

1.2.2 WLAN Positioning with Weighted Graphs

The research project conducted by Hansen and Thomsen (2009), proposes an interesting approach for indoor positioning using the existing Wi-Fi infrastructure of buildings and mobile devices. As with RADAR, this research project applies a fingerprinting approach but takes into account where and how fast the user can physically move.

The system is capable of estimating a position with an accuracy three meters with 90% precision.

This approach is further described in detail in Chapter 5.

1.2.3 BLIP

A BLIP System is a mobile marketing platform that enables companies to define campaigns for the end-users having a Bluetooth enabled device. A campaign could for instance be advertisements that relates to nearby shops. The system consists of BlipNodes, each of which constitutes a specific area of interest. These are essentially Bluetooth access points that enable one-to-one communication with the target audience. The BlipNodes detect presence based on proximity, that is, they register Bluetooth enabled devices that are within reach. Registered data about nearby Bluetooth devices is forwarded to a central server. The BlipNodes can either be wired or wirelessly connected to the central server through the Internet. This opens for a single server controlling multiple BLIP Systems.(BLIP, 2010)

The BLIP System gives inspiration in a variety of ways for BlueCAML. Initially, the idea of centralising configuration of the system to a central server could be adopted profitably such that configuration does not need to be performed in multiple areas of the system. For instance, one could imagine, that in case the building in which BlueCAML is deployed changes environmentally, thus, invalidating the existing environmental information, corrections to these can be created and placed on the central server and pushed onto the mobile devices when they reconnect. Finally, it may be appropriate to adopt the idea of pushing content to the mobile devices through established Bluetooth connection from the sensors instead of using a separate communication path for that such as the use of the 3G network.

1.2.4 Easy Clocking

Last semester the authors developed Easy Clocking (Sw701b, 2009), a system capable of automatically clocking in and out employees according to their position. Bluetooth was used to estimate the range between a fixed point and a Bluetooth tag. The research showed that it was possible to estimate this range with an accuracy of five meters 87% precision in an open corridor and under preferable conditions, using the RSSI as means of deriving a distance estimate (Sw701b, 2009).

Easy Clocking works by deploying location sensors at known positions, whose purpose is to scan the area for Bluetooth tags. When tags are discovered, the measured RSSI value is

used to calculate the distance to them. This information is then stored in the location sensor along with a time stamp. When the tag moves outside the detectable area, the time stamp and distance, along with another time stamp of when the tag was last detected is send to a server.(Sw701b, 2009)

The development of Easy Clocking showed that Bluetooth as an indoor positioning technology holds great potential, and hence established the ground for developing an indoor positioning system using Bluetooth.

1.3 Learning Goals

Besides giving a solution to the previously described scenario, the project must fulfil the goals from the study regulation for a 8^{th} semester software engineering project. To gain knowledge in even more aspects of software engineering, additional goals are set. These are chosen based on what the authors find interesting from previously untried aspects of software engineering related to the development of BlueCAML. The goals from the study regulation and the additional chosen goals are listed in the following:

- Demonstrate knowledge and understanding in analysing, designing, implementing and evaluating software for a mobile platform.
- Consider concepts and opportunities within mobile technologies.
- Account for and assess opportunities in verification and validation of software systems.
- Demonstrate skills in applying techniques, methods, models, and tools for test and verification of software systems.
- Gain knowledge in distributed software development.
- Gain knowledge in using the model checking tool UPPAAL for verification.
- Examine different positioning techniques, implement one or more techniques, and test them in a practical setting.
- Gain knowledge in the capabilities of Bluetooth in regards to indoor positioning systems.

1.4 Report Overview

Even though an iterative and incremental development method has been used, this will not be reflected in the structure of the report. This means that the report will document the final product, and not how it has evolved over time. The authors have good experience in structuring the report this way, and hence want to repeat it.

The following two chapters describe the development method and the requirements for BlueCAML. The remaining report is structured into the following five parts:

Part 1: Analysis The Bluetooth technology, positioning methods, topology of positioning systems, fingerprinting methods, and test and verification methods are analysed and decisions are made based on this.

Part 2: Architecture This part describes the architecture of BlueCAML and how the different components of the system interact.

Part 3: Design This part describes the design of each of the major components of BlueCAML and which subcomponents they consist of.

Part 4: Implementation The implementation part provides concrete implementation details of some of the interesting areas of BlueCAML.

Part 5: Conclusion The project of BlueCAML is concluded, and encountered problems, what have been learned, and possible improvements are finally discussed.

2
Development Method

This chapter describes the development method adopted in the development of BlueCAML.

Generally, the purpose of using a development method is to help structuring and controlling the development process (Sommerville, 1999). The chapter is based on experience gained by using different development methods on previous projects (Weisberg et al., 2007; Frost et al., 2008; Sw701b, 2009). Namely the incremental method (Sommerville, 1999), Object-Oriented Analysis and Design (OOAD) (Munk-Madsen et al., 2000), and a tailored Scrum (Larman, 2003; Sw701b, 2009) have been applied. The tailored Scrum, used for developing Easy Clocking (Sw701b, 2009) used practices and work products, based on prior experience, from various other methods, such as the problem and application domain analysis known from OOAD and the vision work product known from Unified Process (UP).

According to the development of Easy Clocking, using the tailored Scrum development method generally was a success. In this, work products such as user stories, time estimates, and burn down charts turned out beneficial in aiding with assessing the progress of the development process contrasting the experience gained from more traditional methods. However, place for improvements was identified. Therefore, to explore the full potential of the method, the authors want to adjust it further.

An adjustment, is to incorporate the process of writing a report into the Scrum project management, since managing report writing separately was problematic in Easy Clocking. Therefore, user stories with corresponding estimates for the report are needed.

Another lesson learned from Easy Clocking is that iteratively developing software using Scrum requires developers to refactor or add new functionality to unfamiliar source code. Undesirably, this often results in the developers introducing faults to the software (Sw701b, 2009). To counter this, a test-suite with automatic unit tests proved effective in catching newly introduced faults. Based on this experience, more focus is to be put in testing the software using different testing techniques, which also complies with the learning goals.

The following section describes the tailored Scrum development method in detail with the chosen practices and work products to be used after applying the specified changes.

2.1 Tailored Scrum

The authors have previously used or been introduced to various development methods. This provides the basis for selecting practices and work products that through assessment seem most interesting and valuable for the development of the project.

- Backlogs. (Scrum)
- Burn Down Chart. (Scrum)
- Iterative Sprints. (Scrum)
- User Stories. (XP)
- Planning Game. (XP)
- Problem and Application Domain. (OOAD)
- Comprehensive Documentation.

A more detailed description of why the practices and work products are chosen is given in the following sections.

2.1.1 Backlogs (Scrum)

One of the key work products in Scrum is the use of backlogs, namely the product and sprint backlog. The product backlog consists of all features, such as user stories etc., that must be reflected in the development of the product.

In the sprint backlog, a subset of items from the product backlog is chosen to be conducted during the particular sprint. These items are decomposed into tasks each of which can be in one of the three states: Planned, Ongoing and Done. Normally, the items for the sprint backlog are chosen in cooperation between all the stakeholders, but in the development of BlueCAML, only the authors have been involved in this process.(Larman, 2003, c. 7)

Appendix D shows the sprint backlog from the first sprint of BlueCAML.

2.1.2 Burn Down Chart (Scrum)

Using a burn down chart to monitor the progress during a sprint, was a motivating factor for completing tasks during the last semester project. Therefore, this work product will likewise be incorporated into the development process of BlueCAML. The burn down chart is updated as the last thing every afternoon, providing the developers a consistent overview of the progress thereby aiding in assessing whether the sprint backlog can be completed on time or actions need to be taken accordingly.(Larman, 2003, c. 7)

Appendix D shows the burn down chart for the first sprint.

2.1.3 Iterative Sprints (Scrum)

Scrum, and other agile development methods (Larman, 2003, c. 7), encourage the use of incremental and iterative development. In Scrum the iterations are called sprints. The duration of the sprints may vary from project to project. The authors have used sprints of two weeks on a previous project which differs from the normal four week sprints encouraged by Scrum. The sprint length is chosen in order to more quickly adapt to changes in requirements, adapt to additional work related to meetings, and more quickly detect time slippage as a consequence of bad time estimates. The downside of the decreased sprint length is more overhead related to evaluating and planning the sprints.

2.1.4 User Stories (XP)

A user story consists of a sentence and a time estimate indicating the effort for completing it. The purpose of the sentence is to refresh the conversation that led to the creation of the user story. Furthermore, acceptance tests can be added if needed. (Cohn, 2004)

Scrum does not define how items are represented in the product backlog. Therefore, work products from other development methods can be applied for this. The combination of Scrum and usage of user stories, known from XP, has been introduced on a previous project (Sw701b, 2009) where this combination was applied with success. Hence, this work product will also be applied in this development process.

2.1.5 Planning Game (XP)

Scrum encourages giving each user story in the backlogs time estimates (Larman, 2003, c. 7). Last semester, experience was gained in the Poker Planning Game (Atira, 2009) time estimation technique which allows the developers to get insight into the different interpretations of the tasks and gain a shared understanding of them.(Sw701b, 2009)

When using the Poker Planning Game, each developer estimates a given task and compares this with the estimates made by the other developers. Afterwards, the developers with the highest and lowest time estimates, respectively, explain how they derived the specific estimate. Based on this, the activity is repeated and new time estimates are made, until the developers come to an agreement.(Atira, 2009; Sw701b, 2009)

2.1.6 Problem and Application Domain (OOAD)

In previous projects (Frost et al., 2008; Sw701b, 2009), the problem and application domain work products were used. In the development of a privacy module to StreamSpin (Frost et al., 2008), they were used with comprehensive documentation. In the development of Easy Clocking, the work products were only limited used to introduce the two domains for the reader. This turned out to work well and will therefore be repeated.

2.1.7 Comprehensive Documentation

Scrum encourages the use of documentation. However, it spans wide on the ceremony scale meaning that it does not exclude the possibility of comprehensive documentation which is required for an academic project report.

When developing Easy Clocking, the concept of limited documentation was used when implementing the user stories, and then afterwards, as a post-rationalisation, the documentation with corresponding diagrams was made. This was a great advantage since it revealed design flaws etc. which then could be corrected before moving on to the next user story (Sw701b, 2009). Therefore, this will be repeated in the development of BlueCAML.

3 Requirements

The following lists the requirements for BlueCAML divided into functional and non-functional requirements.

The requirements have been categorised into: *Learning goal*, *Scope*, and *User story*. The learning goal requirements are selected in order to fulfil the goals set by the study regulation and to set goals for new subject matter to be learned in this project. The scope requirements are set in order to define the scope of the project. Finally, the user story requirements are defined in order to specify more clearly the goals of the final product.

In the following, both functional and non-functional requirements are given. Furthermore, a set of quality factors are given in the non-functional requirements.

3.1 Functional Requirements

The functional requirements are listed below:

- Estimate current position of users in an indoor environment. (Learning goal)
- The position must be shown on a map of the building. (User story)
- Position estimates must be provided with an interval of at maximum four seconds. (User story)
- The accuracy of the position estimates must at minimum have an accuracy of three meters with 80% precision. (Scope)
- The user must be able to report environmental changes and these must be retrievable by other users. (User story)
- The average walking speed of the users is 1.4 m/s (5 km/h). (Scope)

3.2 Non-functional Requirements

Following are the non-functional requirements for BlueCAML.

- The system must be operational in a 100 square meter area with 10 people. (Scope)
- The system must be operational in indoor environments. (User story)
- The mobile application must be executable on a mobile device. (Learning goal)
- The BlueCAML infrastructure must be comprised of relatively low-cost equipment. (User story)
- The BlueCAML system must be scalable in terms of number of concurrent users. (User story)

Besides these requirements, the quality factors for BlueCAML are also identified. These are described in the following.

3.2.1 Quality Factors

Quality factors are used to express to which degree different aspects for the final product are important for the customer. E.g. besides expecting the software to meet the user requirements, the customer might also want the software to be flexible if the circumstances, in which it is used, are likely to change.(van Vliet, 2008)

Prioritising the quality factors helps with architectural and design decisions throughout the project and helps setting the scope such that various irrelevant issues are stated explicitly not to be taken into account. The quality factors are prioritised in the following categories:

Very Important Quality factors requiring much focus during the development, essentially meaning that the product is primarily based on these.

Important Quality factors requiring focus but not to the same extent as those with a very important prioritisation.

Less Important Quality factors which are not focused on. However, they will be taken into account to the extent of what is good practice.

Irrelevant Quality factors not considered during development.

The definitions of the quality factors can be found in Appendix A. Table 3.1 shows the prioritisation of the quality factors regarding BlueCAML.

Argumentation for the prioritisations is given below.

Quality Factor	Very Important	Important	Less Important	Irrelevant
Product Operations				
Correctness	✓			
Efficiency		✓		
Usability		✓		
Reliability		✓		
Integrity			✓	
Durability				✓
Availability				✓
Product Revision				
Testability	✓			
Scalability		✓		
Flexibility			✓	
Maintainability			✓	
Product Transition				
Reusability			✓	
Portability				✓
Interoperability				✓

Table 3.1: Prioritisation of the quality factors grouped into: *product operation*, *product revision*, and *product transition*.

Very Important

Correctness The final system must satisfy the requirements, since these dictate what functionality is considered applicable for a user. Hence, if not fulfilling these, it is not assumed that the users are satisfied with the system.

Testability This quality factor has not been focused on in previous projects, meaning that focusing on this will provide new knowledge which can be used later. Due to the project taking starting point in mobile development with positioning in mind, focusing on testability is beneficial because changes require setting up the environment which is a time consuming process. Accommodating this can be done by emulating it through simulation. Additionally, the study regulation dictates the necessity of emphasising on this.

Important

Efficiency The main part of BlueCAML is executed on a mobile device with limited resources, hence this must be taken into account to ensure that the software does not require more resources than available. The same applies for the signaltransmitters whom are based on limited resources due to their embedded nature.

Usability One of the purposes of BlueCAML is to make the basis for collaborative maintenance of the system. The hypothesis is that for this to be a success, it must not be a complicated process for the users potentially degrading the incentive for users to do so. Thus, the user interface and the process the user must participate in should be intuitive and easy to conduct.

Reliability The system must be reliable in the sense that it must be able to handle erroneous corrections in a proper way. Specifically, an erroneous correction should not make the system crash or, in a gentler form, present the misreading in form of an incorrect position to the user. Unfortunately, since the position estimates depends on the environment, it is difficult to prevent scenarios where incorrect estimates are given. Furthermore, the time it takes for BlueCAML to calculate a user's position and present it on the mobile device must be within some predefined interval, or else the presented position will be outdated when given to the user.

Scalability The potential user group of an indoor positioning system, like BlueCAML, is considered to be large, meaning that the system must be capable of handling a large amount of concurrent users. If this is not taken into account, components of the system might need to be redesigned or even the whole architecture must be reconsidered to accommodate an unexpected increase of number of users.

Less Important

These quality factors are not focused on during the development. As an example, consider integrity. In practice, this is an issue which should not be neglected when dealing with user information. This especially applies when dealing with user information that potentially may compromise their privacy. However, ensuring integrity is a subject that has to fully incorporated and hence a significant effort is required. This effort is assessed to completely shift focus from developing the actual positioning capabilities of the system and due to these reasons, integrity concerns are omitted.

Irrelevant

Focus is not put in fulfilling these quality factors. For instance, consider the portability quality factor. This is not considered relevant since it is known from previous experience (Sw701b, 2009), that the functionality of accessing the Bluetooth RSSI values is platform dependent. The purpose of BlueCAML is to show whether Bluetooth is applicable in an indoor positioning system, and whether mechanisms can be implemented, allowing the positioning capabilities to be maintained in terms of accuracy. If these purposes can be fulfilled, future work could include porting BlueCAML to other platforms.

Part I

Analysis

4

The Bluetooth Technology

The purpose of this chapter is to analyse the Bluetooth technology to reveal its characteristics and possible complications that must be considered in the development. The analysis is based on the Bluetooth specification (SIG, 2007).

Frequency Hopping

The Bluetooth radio operates in the 2.4 GHz ISM band; the same as Wi-Fi networks. It uses frequency hopping in order to reduce the impact of interference from other wireless communication using the same frequency. Specifically, Bluetooth hops every 625 microseconds between 78 frequencies with 1 MHz intervals above 2.4 GHz. This impacts the time required for discovering visible devices and connecting to devices since the different frequency hops must be searched to synchronise.

Bluetooth Topology

A device can initiate a simple point-to-point connection or connect to multiple devices, thereby forming a piconet topology. The device initiating the connections in a piconet is called the master and all other devices are called slaves. In a piconet, there can exist only one master and up to seven active slaves and 255 parked slaves. Both master and slaves can be part of multiple piconets simultaneously, hence in this project, the individual signaltransmitters can be slaves in several piconets.

It is only possible to measure the RSSI value of the seven active connections and not inactive or parked connections. However, this is not considered a problem in this project since the maximal number of connectable signaltransmitters at a given position is considered to be four at most.

Received Signal Strength Indicator Properties

The Bluetooth specification does not define exactly how the RSSI values are to be measured by the hardware. For instance, the time it takes to measure the value and when the value

is refreshed due to caching, depends on the hardware implementation. An analysis of the measured RSSI values indicates that the mobile device used in BlueCAML caches these between consecutive measurements. Due to the incompleteness of this specification, this must be accounted for in the development.

Power Management

In the specification, it is stated that class 1 Bluetooth devices must implement power control features in order to limit the power usage of slaves connected to a master device (SIG, 2007, page 32). The master device should notify the slave device if it detects a signal which is stronger or weaker than necessary, by prompting the slave to decrease or increase its radio strength for that particular connection. This is a potential problem since the signals are the basis for making position estimates, and hence must be considered fairly constant. It is not stated clearly if this is the responsibility of the software or the hardware in the specification but the authors have not been able to control this functionality from the exposed API. Regardless, it is not considered a problem since practical measurements have not shown to be affected by this behaviour.

Detecting Range

Class 1 Bluetooth devices, which are used in this project, have a theoretical detecting range of 100 meters. However, during the experiments made inside a building in this project, the practical detecting range is closer to 25 meters (SIG, 2007). This means that for a mobile device to detect multiple signaltransmitters, these should be placed each 25 meters, resulting in the mobile device being able to detect at least two signaltransmitters anywhere. This means that one signaltransmitter approximately covers a 1000 square meter area, which according to the requirements, stating that BlueCAML should be applicable of handling 10 concurrent users in a 100 square meter area, in worst case results in 100 concurrent users per signaltransmitter. Since the signaltransmitters do not need to be actively connected to each mobile device simultaneously, it is assessed that the requirement can be fulfilled. It is expected that even more users can use the system concurrently with the only side effect of having longer communication times due to collisions of the Bluetooth signals. This is opposite to piconets where the frequency hops are divided between the devices.

In the scenario in which BlueCAML is tested, it is practically impossible to acquire 100 Bluetooth devices for testing this property. The requirement is therefore upheld through theoretical basis.

5 Position Estimation

To provide the estimated position of the user on a map, a positioning estimation method is needed. The purpose of this chapter is to highlight the position estimation methods that have been considered as candidates for BlueCAML. The methods are described and their respective advantages and disadvantages relevant in terms of BlueCAML are exposed. This creates the foundation for the argumentation in the subsequent selection of positioning estimation method. Finally, considerations regarding the method with potential additions to both improve accuracy, precision, and search performance are revealed. Accuracy and precision are two common performance metrics of a positioning system whose definitions are described below:

Accuracy Accuracy is a concept used to describe the order the estimated position deviates from the actual position.

Precision Precision refers to the percentage of measurements retaining a particular accuracy.

5.1 Positioning Approaches

The following describes three approaches for estimating positions and is inspired by Liu et al. (2007).

5.1.1 Fingerprinting

Using fingerprints for position estimation refers to the type of approach in which prior measurements of the environment are collected and stored. The RSSI values of detectable signal-transmitters can be used as the measurable entity, and the process of collecting these is referred to as the offline stage. The measurements are called fingerprints, and these are made at selected points in the environment. The fingerprints are collectively stored in a radio map which, besides containing the fingerprints, also contain the corresponding position from which the they were collected.

The offline stage creates the foundation for the online stage where a detectable object is placed in the environment. The object similarly measures the RSSI values of the detectable

signaltransmitters thereby creating a fingerprint. This fingerprint is then compared with the fingerprints stored in the radio map by utilising one or more of the various pattern matching techniques available. The fingerprint of the radio map that best matches the collected fingerprint is chosen and the corresponding position returned. Depending on the used technique, multiple fingerprints may be found as best matching, and from these, a position is estimated.

Advantages

- Relatively good accuracy is achievable provided that fingerprints are available for small distance intervals.
- The fingerprinting approach takes static elements in the environment into account, such as walls, by measuring the changes introduced by these. Thus, the accuracy level of the method is not affected as long as these elements are not moved or changed. This means that the fingerprinting technique is not as vulnerable to radio signal phenomena such as multi-path fading, diffraction, and reflection as other approaches because these are accounted for in the fingerprints during the offline stage.

Disadvantages

- Requires initialising the system by conducting the aforementioned offline stage which may be time consuming.
- Due to environmental changes such as the density of people in the environment and refurnishment, the offline stage may need to be conducted regularly to prevent erroneous position estimates.
- There exist various methods of conducting the pattern matching process, but in some cases, this may be a complex task. The amount of time required to determine the best-matching fingerprint may be a problem if the radio map is very large.

5.1.2 Triangulation

Triangulation is used to estimate a position based on measurements from three signaltransmitters using geometrical views. The geometrical views are either to use lateration or angulation algorithms. The distances to multiple signaltransmitters are used in lateration and in angulation the angles to the signaltransmitters are used (Liu et al., 2007). Figure 5.1 depicts the ideas of both lateration and angulation.

Provided that the position is to be estimated in the plane, knowing the distance to at least three signaltransmitters is sufficient due to a single intersection point being available for the three circles drawn with radii equal their respective distances and origo localised at the signaltransmitters.

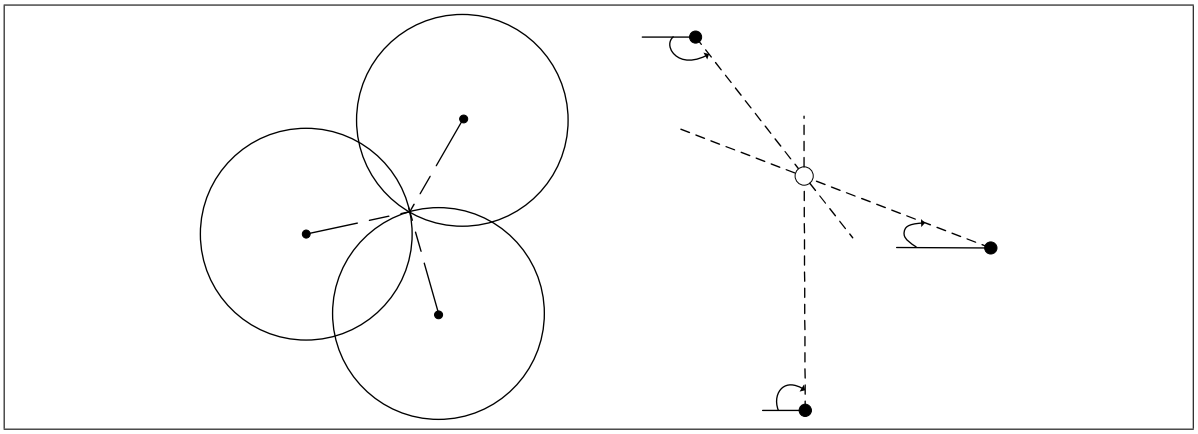


Figure 5.1: Lateration is shown at the left where the distances to three signaltransmitters are known. At the right, the concept of angulation is depicted, using three directions to estimate an intersection point.

Liu et al. (2007) and Malik (2009) describe Time Of Arrival (TOA), Time Difference Of Arrival (TDOA), Received Signal Strength Indicator (RSSI) and Angle Of Arrival (AOA) as techniques for performing lateration and angulation.

TOA Uses the time the signal travels between two devices to estimate the distance. This requires that the clocks of the used devices are precisely synchronised and that the signal contains a timestamp about when it left the source.

TDOA Uses the time difference of the transmitted signal received at multiple sources to estimate the position of the device. This requires that the signaltransmitters are precisely synchronised but there is no requirements for synchronisation on the mobile device.

RSSI Is used to measure the decrease in a signal's strength as it propagates from source to destination. By using a radio propagation model, it is possible to estimate the distance to a signaltransmitter based on this value. The authors applied this technique in previous research (Sw701b, 2009) and showed that an accuracy of five meters with 87% precision is possible when estimating distances under favourable conditions.

AOA Using AOA, a position is estimated using the angle from which the signal is received. Having at least three angles, an intersection point and hence position, can be estimated. This requires that the hardware supports measuring angles.

Advantages

- In respect to the other positioning algorithms, triangulation only requires relatively few computations for position estimation. This is advantageous if the computations are made on a mobile device.

- The triangulation approach generally does not require a comprehensive offline stage. Only the algorithms based on a radio propagation model require calibration after deployment.

- Besides knowing the position of the signaltransmitters, no other data storage is required, thereby being advantageous if the data needs to be stored on a mobile device.

- The triangulation algorithms hold potential for estimating precise positions which are not constrained by some predefined granularity.

Disadvantages

- The techniques using time stamps and angles, require special-purpose hardware such that these can be measured precisely. Time synchronisation is paramount because the signal can travel 300 meters in only one microsecond.(Bose and Foh, 2007; Fischer et al., 2004)

- The algorithms are vulnerable to radio propagation phenomena like multipath fading, which may result in signals propagating in an indirect line from transmitter to receiver.

- If the environment changes significantly, the radio propagation model must be recalibrated.

5.1.3 Proximity

Proximity is a simple positioning approach aimed at providing symbolic relative positions, that is, the system is only capable of informing about whether or not a mobile device is within the detectable range of the signaltransmitter. Therefore, a system based on the proximity approach consists of setting up signaltransmitters at each position of interest. In the case where two or more signaltransmitters observe the same signal, the mobile device is considered collocated with the signaltransmitter that received the strongest signal.(Liu et al., 2007)

Advantages

- Because this approach mainly reports positions based on mobile devices being discoverable, proximity is not significantly affected by the environment such as signals attenuated by walls etc. Therefore, this approach is more robust than the others.

- It does not require heavy calculations and processing of data. Only in case multiple signal receivers receive the same signal, a comparison is needed of the measured RSSI values to estimate the position of the mobile device.

Disadvantages

- If the proximity approach is deployed with the purpose of tracking objects with high resolution, it may require that several signaltransmitters are installed in the environment.
- The accuracy of the proximity approach is constrained to the number of signaltransmitters installed in the environment.

5.1.4 Selection of Positioning Approach

Several criteria are important for the selection of positioning approach. These are cost of deployment, accuracy and precision, need for special hardware, required computational power, and effort required for maintaining the positioning system.

The cost of deployment is considered equal for triangulation and fingerprinting since both approaches require multiple signaltransmitters to be detectable at all positions in the given area. For proximity, the amount of required signaltransmitters depends on the desired accuracy. With an accuracy of three meters, signaltransmitters must be deployed for every six meters, giving it a relatively high deployment cost. The advantage, however, is that good accuracy is achievable. Depending on the requirements for the positioning system, this approach could be either suitable or too expensive in deployment costs. The goal of this project is to develop a positioning system which has a cheaper deployment cost in terms of amount of needed hardware.

In respect to triangulation, the achievable accuracy in environments containing obstacles, resulting in multipath fading etc., is not considered sufficient for presenting the estimated position to a user. This is based on the fact that BlueCAML is using consumer-grade products for signaltransmitters and hence cannot take the advantage of using the time based techniques. Also, previous research, among others made by the authors, revealed that applying the theory of radio propagation is difficult since the relationship between RSSI and distance do not take into account variance in the environment such as density of obstacles (Sw701b, 2009; Bose and Foh, 2007). This is generally overcome using fingerprinting. Even though fingerprinting requires more maintenance and computational power than triangulation, the potential of high accuracy overrules these disadvantages since this is crucial for an indoor positioning system aiming at estimating positions of movable objects.

Research has shown that computations for fingerprinting can be performed on mobile devices (Hansen and Thomsen, 2009). If the computations are made server-side, other problems, like scalability, must be considered where triangulation and fingerprinting, give rise to the same problems.

The fingerprinting approach is therefore used in BlueCAML. In respect to maintaining the system, effort must be put in finding an easy way to do this, because otherwise, it will potentially require too many resources to be affordable for institutions deploying the system.

In the following, a *fingerprint* denotes a fingerprint present in the radio map and an *online fingerprint* denotes a fingerprint measured by the mobile device in the online stage.

5.2 Fingerprinting Techniques

Having selected the positioning approach, the purpose of this section is to uncover the various techniques that can be applied within fingerprinting. The techniques address issues such as reducing the search space of the radio map and how estimation of a position can be performed given an online fingerprint and the radio map. The description of these end up in a comparative analysis in which choices of techniques to be applied will be given.

5.2.1 Weighted Graphs

The principles of this technique is based on the material described in Hansen and Thomsen (2009).

Weighted graphs can be used as a technique to improve the accuracy and computational efficiency of fingerprinting. The concept is to only consider those fingerprints that are feasibly reached from the previous position. This way, errors like when a user is estimated to travel through walls can be reduced and the search space can be decreased, thereby improving both the accuracy and efficiency in terms of searching the radio map.

After the offline stage, the fingerprint positions are modelled as nodes connected by weighted edges to neighbour positions. The weight represents the distance between the two positions. The feasible positions can then be found from a previous position using a breadth first search, searching in a range of how far the user is estimated to have moved. This is called searching in the primary search space.

To prevent situations where the system gets stuck, the technique uses so-called radius nodes, to allow illogical movement if great evidence indicates this. For instance, the system can get stuck if at some point it estimates that the user have entered a room when the user have not. The subsequent collection of fingerprints will be radius nodes and hence indicate that an illogical jump through the wall is necessary to get out of the room. Each node is therefore provided a list of radius nodes which can be reached through illogical movement. The list of radius nodes consists of the nodes that lie within the perimeter of the circle with radius equal to the distance for which no user is expected to have gone further between consecutive position estimates. However, this list excludes the nodes from primary search space. For each position estimate, this list is then, along with the primary search space, searched through. This is called a search in the secondary search space. It is suggested that if two to three consecutive estimates indicate that the user has made an illogical movement, a shift to an illogical position should be made.

Figure 5.2 depicts the Weighted Graphs technique applied to a radio map. The black mark indicates the current position, and the darker grey marks are the nodes that can be reached through logical movement, hence, these compose the primary search space. Finally, the lighter grey marks are the radius nodes used for the secondary search space.

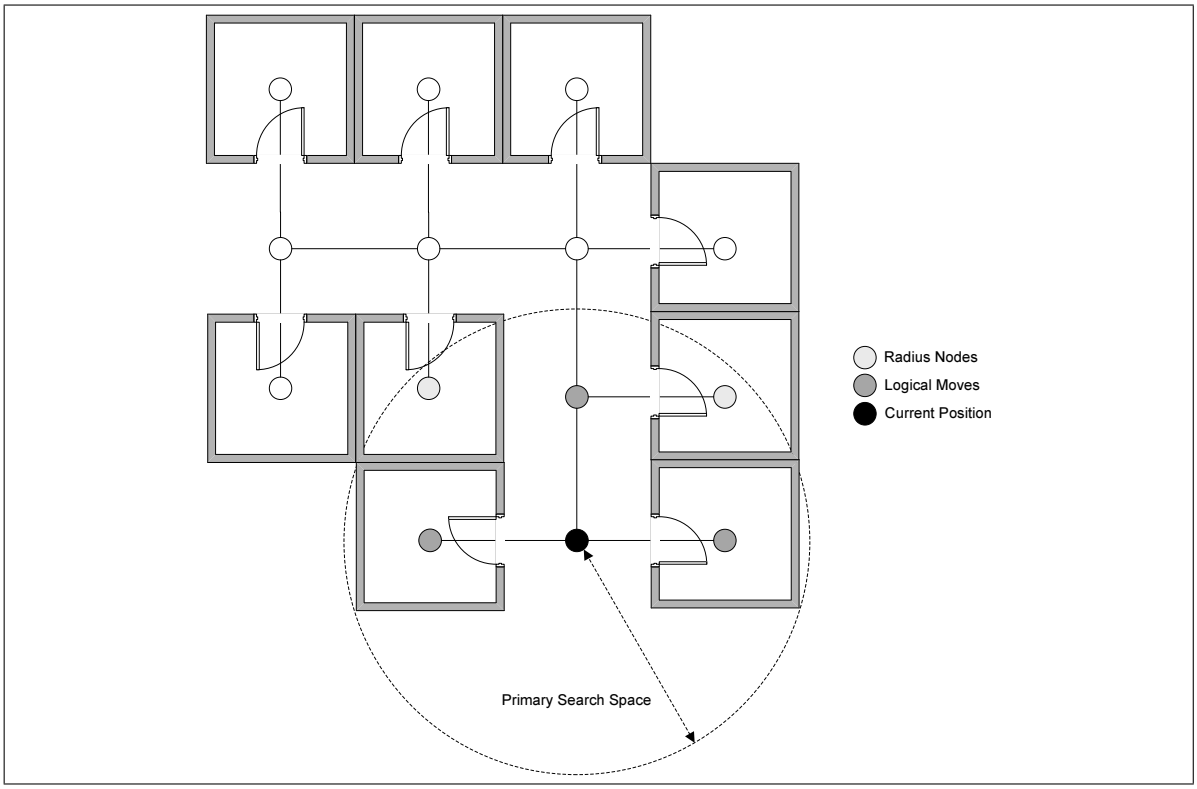


Figure 5.2: Example of using Weighted Graphs to define logical paths between the different fingerprint positions.

5.2.2 Nearest Neighbour

Nearest Neighbour (NN) is one of the deterministic techniques that can be applied for estimating the position of a mobile device using the radio map and an online fingerprint.

For describing the procedure of NN, assume that each fingerprint is represented as an n-dimensional point, a, where each number, $a_1, a_2, \ldots, a_n$, represents an RSSI value of n signaltransmitters. Given the radio map and the online fingerprint, the procedure of NN is to sequentially calculate the distance according to some norm from the online fingerprint to all the fingerprints kept in the radio map. The position of the mobile device is then estimated to be the corresponding position of the fingerprint in the radio map having the smallest distance.(Honkavirta et al., 2009)

There exist various norms which can be used for calculating the distance. Following, is a brief description of two of these:

Euclidean distance The ordinary distance between two points which can be measured with a ruler. Given two points $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$ in n-space, the Euclidean distance between them can be calculated as

$$d(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}. \tag{5.1}$$

Manhattan distance This type of distance between two points $a = (a_1, a_2, \cdots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$ is defined as the sum of the absolute differences of their coordinates. Specifically, the Manhattan distance can be described by the mathematical formula

$$d(a, b) = \sum_{i=1}^{n} |a_i - b_i|. \tag{5.2}$$

Graphically, this means that the Manhattan distance is a grid-like path from a to b whereas the Euclidean distance is a straight line. According to the research conducted by Li et al. (2005), the Manhattan distance yields the best result however it is not significantly better. Due to these conclusions, the Manhattan distance is used for calculating the distance between the two fingerprints.

A problem may occur in case two fingerprints are of different dimensions, that is the addition or absence of RSSI values from signaltransmitters in the online fingerprint compared to the offline fingerprint. For resolving this issue, a penalty mechanism can be implemented as is done in the research project by Hansen (2010). The penalty system works by assigning a default RSSI value to signaltransmitters not represented in the offline fingerprint. In the previously mentioned research project, this default value was defined as the lowest possible RSSI value which is -255.

NN is only capable of estimating the position of a mobile device to be one of the positions of the fingerprints stored in the radio map. Often, a better approach is to make use of K-Nearest Neighbour (KNN) or even Weighted K-Nearest Neighbour (WKNN). KNN is the NN version which chooses the K nearest neighbours to the object of interest. According to Li et al. (2007), using $K = 3$ or $K = 4$ has shown to be good parameters.

In relation to position estimation with fingerprints, KNN can be used to estimate a position of the mobile device instead of simply considering it collocated with the position of a fingerprint as in NN. By determining the K nearest neighbours to the mobile device, one can assign the position of the mobile device to be the average of the K nearest neighbours.

An addition to KNN is to assign weights to all the fingerprints in the radio map, thereby describing WKNN. The procedure is the same as in KNN, however, the impact of fingerprint $1, 2, \cdots, K$ in the calculation of the average decrease according to the weight scheme used. Mathematically, WKNN is defined as

$$\hat{x} = \sum_{i=1}^{M} \frac{w_i \cdot x_i}{\sum_{j=1}^{M} w_j}, \tag{5.3}$$

where $\hat{x}$ denotes the new position estimate, w denotes an assigned non-negative weight and, finally, x denotes a point representation of a fingerprint in the radio map (Honkavirta et al., 2009). According to Li et al. (2007), one possible weight scheme is to calculate the inverse of the norm, such as the Euclidean distance, from the RSSI measurement to the particular fingerprint in the radio map. This means that if the Euclidean distance is low, it implies a high weight and opposite. Note, that if equal weights are assigned each fingerprint, the formula is the same as KNN.

5.2.3 Probabilistic Estimation

An alternative to the deterministic methods is to store the different measured RSSI values, e.g. in a histogram, and then use a probabilistic approach to estimate positions. This means that a fingerprint in the radio map consists of histograms of RSSI values for each detectable signaltransmitter. According to Li et al. (2007), Bayes rule can be used for this in the following way

$$P(L_t|O_t) = \frac{P(O_t|L_t) \cdot P(L_t)}{P(O_t)}, \tag{5.4}$$

where L_t is a position at time t, O_t is an observation of RSSI values at time t, and $P(O_t)$ is a normalisation factor.(Li et al., 2007; Fenton, 2010)

$P(O_t|L_t)$ can be determined using histograms of RSSI values of the discoverable signaltransmitter for each position, since these describe the likelihood of an observation at a specific position. This means that given a collected RSSI value, this is compared against the corresponding histogram and the percentage of the occurrence of that RSSI value is then the result. $P(L_t)$ can be set equally for each position, hence, assuming that the likelihood of being at each position is the same. Another scheme is to initially assign equal values of $P(L_t)$ when the user enters the environment. Then, as the user moves around, the value of $P(L_t)$ at each position can be adjusted to take into account historical data such as where the user was previously located.(Li et al., 2007)

Notice that $P(O_t)$ can be determined using marginalisation, which is defined(Fenton, 2010) by

$$P(O_t) = \sum_i P(O_t|L_i) \cdot P(L_i). \tag{5.5}$$

Li et al. (2007) describes that their research showed that using the probabilistic technique gave better results than using NN.

5.2.4 Selection of Fingerprinting Techniques

Above, a number of different techniques for estimating the best matching fingerprint is given. The Weighted Graphs technique can be used in conjunction with NN and the probabilistic techniques, and is hence applied to reduce search space and improve accuracy.

Even though research indicates that the probabilistic techniques yield better results than NN, it has been decided not to use it in favour of NN. As mentioned earlier, Hansen and Thomsen (2009) has used Weighted Graphs and in addition uses KNN and achieves an accuracy of three meters. Therefore, given the relatively more simplistic nature of KNN together with the fact that sufficiently high accuracy has been achieved with it, this fingerprinting technique is used.

6 Positioning Topology

Before analysing and determining the architecture of BlueCAML, the positioning topology needs to be decided upon. The following section analyses the different topologies available for positioning systems and concludes which one is the most suitable for addressing the scenario of BlueCAML.

6.1 Topologies

The position of the mobile devices is estimated by measuring RSSI values as described in Chapter 5. The measurements can be conducted by either the mobile devices or the signal-transmitters and are further processed by either a back-end or on the mobile device.

Four different topologies for placing the positioning logic are been defined by Liu et al. (2007). The following discusses the advantages and disadvantages introduced by these.

To maintain overview for the reader, all advantages and disadvantages for the topologies are described even though some may overlap.

6.1.1 Remote Positioning System

In this topology, the signaltransmitters measure the RSSI values from the mobile devices and pass this information to a back-end which calculates the position of them.

Advantages

- The radio maps only need to be maintained on the back-end since none of this logic is placed on the mobile devices.
- It does not require specific hardware-requirements on the mobile device in relation to position estimation because computations in this regard are placed on the back-end.
- Promotes platform-independence on the mobile device because it does not require communicating with the incorporated Bluetooth adapter.

Disadvantages

- Unable to present the estimated position to the user. This is a crucial disadvantage since this is the specific purpose of BlueCAML.
- Scalability, with respect to number of concurrent users, is not promoted in this topology because the computational effort required for position estimation linearly increases as the number of users increases, which is a disadvantage in respect to some of the other topologies.
- Using specific features of the mobile device to aid position estimation, such as a compass, requires that this information is transmitted to the back-end.
- The user must trust the BlueCAML provider not to misuse the position estimates, by e.g. selling the information to a third-party without the user's knowledge.

6.1.2 Self-Positioning System

In this topology the mobile device uses RSSI values from the signaltransmitters to calculate its own position.

Advantages

- The topology promotes scalability with respect to the manageable number of users because it does not require additional computational power of the infrastructure. This is because the computations for position estimates are placed on the mobile devices themselves and thus there is no need to add more computational power to the infrastructure in order to support more mobile devices.
- Data from sensors on the mobile device, such as a digital compass, can be taken into account when estimating positions without requiring a new communication path.

Disadvantages

- The purpose of the system is limited to functionality similar to navigation, since the position is not given to any external parties, which can provide context-aware information.
- Information about the environment is required to reside on the mobile device for enabling it to estimate its position. In relation to fingerprinting, this includes the radio map.
- Computations are made on a mobile device with limited resources, possibly limiting the complexity of tasks that can be conducted within a given time frame.

- The radio maps must be distributed to the mobile devices, and updates to them must be propagated.

- Complications can occur if the mobile application should be supported by multiple platforms since it requires the mobile devices to expose an API for reading RSSI values from Bluetooth adapters.

- Environmental information, such as people density, from the signaltransmitters cannot be used in the position estimates without establishing a new communication path.

6.1.3 Indirect Remote Positioning System

If the estimated position from a Self-Positioning System is transmitted to a back-end, this is called an Indirect Remote Positioning System. Hence, it is to some extent similar to a Self-Positioning System topology and, hence, some of the advantages and disadvantages are the same.

Advantages

- An increase in the amount of mobile devices using BlueCAML does not require additional computational power of the infrastructure. Only the network traffic is increased due to position estimates need to be transmitted to the back-end.

- BlueCAML can be extended to make use of the position estimates on the back-end to e.g. provide context-aware information to the user.

- The mobile device can take into account its own features, such as a digital compass, when estimating its position without introducing a new communication path.

- The back-end can combine the received position estimates with environmental data received from the signaltransmitters. As described in Chapter 7, this can e.g. be used to help maintaining the radio maps.

- The topology promotes scalability with respect to the manageable number of users because it does not require additional computational power of the infrastructure. This is because the computations for position estimates are placed on the mobile devices themselves and thus there is no need to add more computational power to the infrastructure in order to support more mobile devices.

- Data from sensors on the mobile device, such as a digital compass, can be taken into account when estimating positions without requiring a new communication path.

Disadvantages

- The purpose of the system is limited to functionality similar to navigation, since the position is not given to any external parties, which can provide context-aware information.
- The positions must continuously be transmitted to a back-end. Only doing this in intervals would reduce the disadvantage.
- Complications can occur if the mobile application should be supported by multiple platforms since it requires the mobile devices to expose an API for reading RSSI values from Bluetooth adapters.
- Computations are made on a mobile device with limited resources. The user must trust the BlueCAML provider not to misuse the position estimates, by e.g. selling the information to a third-party without the user's knowledge.
- Information about the environment is required to reside on the mobile device for enabling it to estimate its position. In relation to fingerprinting, this includes the radio map.
- Environmental information, such as people density, from the signaltransmitters cannot be used in the position estimates without establishing a new communication path.
- The radio maps must be distributed to the mobile devices, and updates to them must be propagated.

6.1.4 Indirect Self-Positioning System

This topology is to some extent similar to an Indirect Remote Positioning System. However, the position is estimated by the back-end and transmitted to the mobile device.

Advantages

- This topology does not require the mobile device to do computations.
- The radio maps only need to be maintained on the back-end since no logic is placed on the mobile devices.
- BlueCAML can be extended to provide information such as context-aware information, to the user.
- The mobile application does not require communicating with the underlying Bluetooth component, hence a more portable client can be made.

Disadvantages

- Information from internal sensors on the mobile device is not directly accessible.
- The load of the back-end will increase linearly with the amount of mobile devices, which is a disadvantage in respect to some of the other topologies.
- The user must trust the BlueCAML provider not to misuse the position estimates, by e.g. selling the information to a third-party without the user's knowledge.

6.2 Selection of Topology

The Self-Positioning System, Indirect Remote Positioning System and Indirect Self-Positioning System can all be used as topologies in BlueCAML. The Remote Positioning System topology cannot be used, since the user cannot be presented with the estimated positions.

It has been decided to use the Self-Positioning System topology. This decision is primarily based on, that this topology promotes scalability in terms of manageable number of concurrent users and hence there will be no need to extend the infrastructure with additional computational power in case the user base increases. This is in accordance with the scalability quality factor which is prioritised important. In this decision portability concerns have been left out due the prioritisation of the portability quality factor.

A problem, however, with the Self-Positioning System topology is that a mechanism or procedure must be established to enable that corrections to the radio map submitted by other users get propagated to the remaining mobile devices accordingly. This can either be done manually or automatically.

To solve the problem a communication path needs implemented. An obvious solution would be to make use of the Bluetooth infrastructure constituted by the signaltransmitters. Alternatively, the mobile device and the back-end could communicate through the GPRS network, but it has been assessed that it is better to take the advantage of having the Bluetooth infrastructure at disposal. The signaltransmitters are a link between the back-end and the mobile device. Furthermore, this communication path opens for the opportunity, of relatively simple extending BlueCAML in the future with tracking capabilities where the signaltransmitters detect devices in a given area and report this to a back-end. In previous research (Sw701b, 2009), it was showed, that the infrastructure can be used for this tracking purpose.

7

Environmental Factors Influencing Fingerprinting

The purpose of this chapter is to describe different concerns regarding the offline and online stages of fingerprinting.

The basis of the considerations originates from other research describing various observed phenomena when using fingerprinting (Bahl and Padmanabhan, 2000; Sw701b, 2009; Yeh et al., 2009; King et al., 2006; Lionel M. Ni and Patil, 2004; Hansen and Thomsen, 2009). To clarify whether or not the observed phenomena are actual problems that must be accounted for, experiments have been conducted. Specifically, RSSI measurement method, user orientation, radio map becoming obsolete over time, people density, changes in measured RSSI values over time, and density of fingerprints are investigated. Finally, time consumption of conducting a fingerprint is investigated and a strategy for fingerprinting is given. In this regard, observations and concerns involved in RSSI measurements from the used hardware are given.

This chapter also acts as an indicator of the possibilities in using Bluetooth as a positioning technology.

7.1 RSSI Measurement Method

The RSSI values are provided by the hardware through standard API calls defined by the Bluetooth specification. Unfortunately, a lot of behaviour regarding RSSI values, such as the granularity of the RSSI values and the time interval between updates of them, are left to be decided by the hardware vendors. Due to the loosely defined nature of the Bluetooth specification, this consequently means, that Bluetooth devices from distinct vendors, are likely to behave differently in terms of retrieved RSSI values resulting in being incomparable.(SIG, 2007)

The implementation of RSSI is not documented for the hardware used in this project. However, through observations, it is determined that the implementation updates an internal RSSI value cache whenever it receives a signal in the form of a packet from one of its peers, and it is this value which is returned when enquired. If multiple enquiries are made for the RSSI value in-between updates, the same value is returned multiple times. The authors observed this by studying a list of consecutive RSSI measurements, where a pattern was identified.

Another implication of the internal cache which was observed, is that if a Bluetooth device is continuously enquired for RSSI values and the connection is lost meanwhile due to for instance the user moving out of detecting range, the hardware will continue to return the RSSI value which was last placed in the cache until a connection time out. This time out is observed to be around 20 seconds, is exceeded.

To ensure that a retrieved RSSI value has not previously been retrieved it is therefore necessary to actively request a signal from the signaltransmitter. This can be done using a variety of methods of which the following are considered most appropriate:

L2CAP Ping A solution is to use the ping capabilities of the Bluetooth transport layer, L2CAP. This layer controls the individual connections between the devices and provides a low-level communication between them.

This approach results in a relatively low overhead since the L2CAP ping is designed for availability checks between devices. Experiments on other hardware than the hardware used in BlueCAML shows that a ping and subsequent RSSI measurement can be done in approximately 12 milliseconds.

L2CAP Services As an alternative, some L2CAP level services are made available through the Bluetooth API such as a service for remote device name retrieval and querying for available Bluetooth services. Initial experiments with these methods showed availability checks with subsequently RSSI measurement times in the order of 40 milliseconds.

RFCOMM Ping The final solution is to initiate an RFCOMM connection, which can be compared to a TCP socket connection, between the two devices. Through this, it is fairly simple to transmit a small packet to a receiver which, when receiving the packet responds, thereby simulating a ping request. A substantial drawback of this approach is the necessity of an application running on the signaltransmitters accepting socket connections, which in turn increases the requirements for logic on the signaltransmitters. Initial experiments showed that this approach allows for a ping and subsequent retrieval of an RSSI value in 12 milliseconds.

7.1.1 Conclusion

Initially, the L2CAP ping method would seem as a good candidate since it does not require additional logic on the signaltransmitters in contrast to the RFCOMM ping method, and it allows for a faster retrieval of RSSI values than using L2CAP services. Unfortunately, this method is not able to be implemented on the mobile device software used in this project since the platform does not expose an API allowing direct access to the L2CAP layer.

The choice is then left between RFCOMM ping and L2CAP services. The absence of required logic on all signaltransmitters is appealing since it opens for the possibility of having two types of them, designated as smart, and dumb signaltransmitters, respectively. The incentive for distinguishing between two different types of devices is that it potentially reduces the cost of deploying the system since one could imagine that only one smart signaltransmitter

is required to be discoverable at any time in the environment and the rest can be dumb ones. Dumb signaltransmitters could for instance be made of an inexpensive computer without WLAN capabilities or even a special purpose low-powered device with Bluetooth support.

The RFCOMM ping method, however, allows for more RSSI measurements under the time constraints but as will be shown later, it is determined that the fewer RSSI values measured using the L2CAP remote name query are sufficient for position estimation. Therefore, there is no incentive for using the RFCOMM ping method and hence the L2CAP services method is used in order to solve the problem and for providing a means of having two types of signaltransmitters.

7.2 Orientation of User

The RADAR (Bahl and Padmanabhan, 2000) research project describes, that the human body itself is a factor which may cause signal fluctuations because the signal attenuates or is absorbed when propagating through it. This is a problem which arises often because the movement direction of a user may require the signal to propagate through the body.

For investigating the issue further and determine whether it poses an actual problem, an experiment is made. In the experiment, three signaltransmitters, ST1, ST2, and ST3, are placed in a room, as shown in Figure 7.1. A mobile device is placed in the middle of the room, collecting RSSI values from each signaltransmitter. In the experiment, a fingerprint is made when facing north, south, east, and west. A significant change in the fingerprints indicates that the orientation affects the fingerprints.

The general approach used in the experiment is to collect 100 measurements which are averaged from each signaltransmitter ten times. Previous experience has shown, that using this method gives a representative picture of the environment, both with regards to the number of measurements and to the fact that the same experiment is repeated ten times.(Sw701b, 2009; Bose and Foh, 2007; Malekpour et al., 2008)

Table 7.1 summarises the results from the experiment.

Orientation	ST1	ST2	ST3
North	**-8.2**	**-1.7**	-3.5
South	**-4.9**	**-6.2**	-0.4
East	-6.3	-5.8	-1.2
West	-5.6	-3.6	-8.7

Table 7.1: Average measured RSSI values when facing different directions.

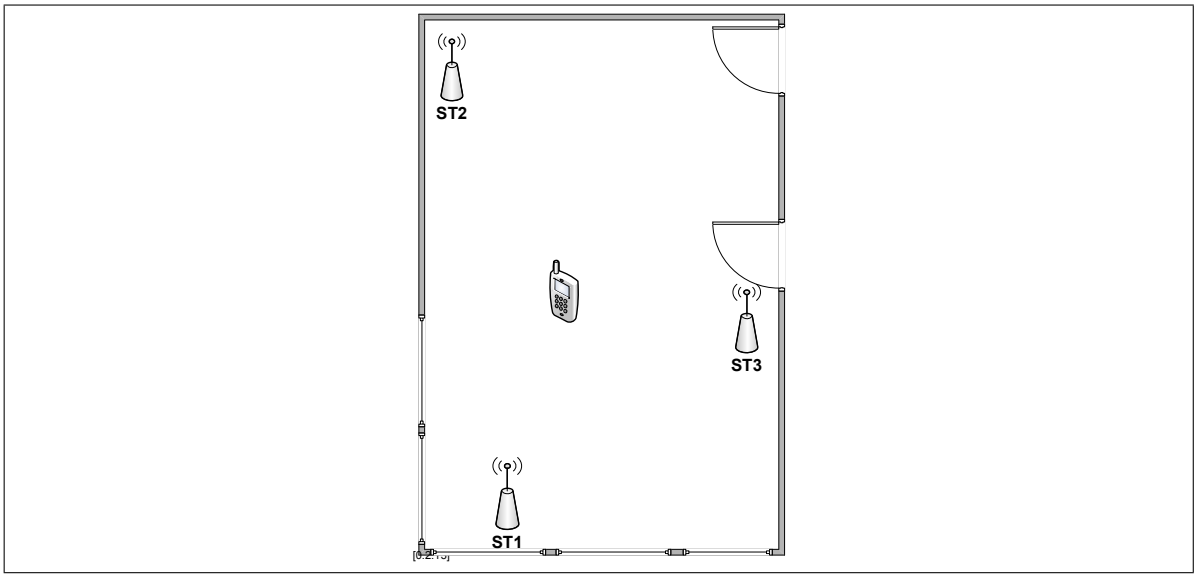


Figure 7.1: The setup used for the experiment. Three signaltransmitters, ST1, ST2, and ST3, are placed in the room, and a mobile device measures the RSSI values from each of them. ST1 and ST2 are placed in south and north, respectively.

7.2.1 Conclusion

As shown, the orientation does influence the measured RSSI values. As an example, consider the difference in average RSSI value for ST1 and ST2 when facing north and south, respectively. When facing north, the RSSI value of ST2 is lower than ST1 and when facing south, the opposite is observed. A similar observation can be made for ST3 when facing east and west.

To accommodate this issue, the radio map can be constructed by taking into account the orientation of the user. As an example this can be done during the offline stage by collecting fingerprints in orientations corresponding to the four corners of the world or, given that the particular rooms are made up of rectangles, a wall can be used as reference (Bahl and Padmanabhan, 2000). Alternatively, the fingerprint could be made by turning around with constant velocity in a particular period of time thereby measuring the average RSSI values for all orientations. It is assessed, that given that this action has to be performed by the users of the system, the former approach where four specific orientations are used as reference will give the best result. The reason for not choosing the rotation option, is the requirement for rotating with constant velocity and also, one could imagine that the user will spend more time in the initial and end position.

The advantage of using the four corners of the world is that the system can be generalised to be used in environments with other shapes than squares and hence target a larger market. A great disadvantage of this approach is that it requires that the users know their orientation with respect to the four corners of the world. However, currently some mobile devices contain an built-in digital compass that could aid the user in knowing her orientation but they are

not common and hence it cannot be assumed that it is available for all users. However, given that a compass is standardised, this option would be preferable (King et al., 2006). The best option is assessed to be using the four corners of the world as reference.

When the four fingerprints are collected, a choice has to made whether to store the four fingerprints in the radio map and associate them with the particular position from which they were collected or to average the four fingerprints and associate that single fingerprint with the position. If the four fingerprints are stored separately in the radio map, it requires four times as many searches when matching a fingerprint, hence increase resource usage. Therefore, the more simple solution of storing the average of the four fingerprints is used.

7.3 Radio Map Becoming Obsolete Over Time

In previous research (Sw701b, 2009), it was experienced that the propagation of radio signals is highly dependent on the environment. In that connection, a problem is that the radio map needs to be maintained as a result of environmental changes, for instance due to refurnishing.

To verify the presence of this problem, the following experiment is made using the same scenario as the experiment in Section 7.2. First, a fingerprint is made in a room. Afterwards, the room is refurnished and a new fingerprint is made. A change in the fingerprints then indicates how much refurnishing influences the radio signals.

The results of the experiment are shown in Table 7.2.

Refurnishing	ST1	ST2	ST3
Before	-4.1	**-8.2**	-2.7
After	-2.6	**-4.6**	-1.0

Table 7.2: Summary of the measurements made before and after refurnishing.

7.3.1 Conclusion

As shown, refurnishing affects the measured RSSI values. Especially, notice the average RSSI value measured from ST2, which almost halves.

To accommodate the changes in the environment over time, the radio map needs to be calibrated. The recalibration process is expected to be too time consuming if the responsibility is placed on a small number of maintainers dedicated for this purpose. Instead, another approach could be to place this responsibility on the users of BlueCAML. This is an interesting approach because, provided that the users are willing to spend time improving the system for their own benefice, maintenance of BlueCAML can be distributed system. A distributed system, is defined as a system where each participant helps in achieving the common goal.

However, placing the responsibility of maintenance to the users give rise to other problems mainly regarding usability and the correctness of the collected information. If a user is given the task of maintaining the radio map, a mechanism must be implemented to ensure that incorrect fingerprints are not placed in the radio map. A solution to this problem can be analogous to how Wikipedia prevents articles from being vandalised due to its open nature. Wikipedia accommodates this problem by assigning roles to different users such as administrators, and unregistered contributors (Wikipedia, 2010). These have individual privileges in regards to modifying the content of the articles.

A variant of the Wikipedia approach would be to have two types of fingerprints in the system: official fingerprints and user corrected fingerprints. The official fingerprints are conducted by people who are entrusted to conduct correct measurements, both in terms of working equipment and correct location and direction awareness. Corrections conducted by the users of the system when maintaining the system, are not considered as trustful as the officials since the user might have made an error.

The basic principle is therefore to return the newest official fingerprint. Since this does not take into account the user generated correction, it can be overruled by two consecutive corrections approximately equal. Such two consecutive corrections would strongly indicate that the environment has changed, and an average of the two is returned. This procedure is used in BlueCAML.

7.4 Varying People Density

Another issue is signal attenuation caused by human bodies in the vicinity of the signaltransmitters and mobile devices. For instance, suppose that positioning is to be deployed in a canteen. During lunch hours, the canteen may be crowded and during the rest of the day, close to empty. The same example holds for open office spaces that may contain few people at the end of the day.

The factor of people density is, presumably, of importance if high accuracy is desired during the entire day. Similar to the experiment described in Section 7.2, measurements are made to verify the existence of the problem. Again, 100 measurements are collected from each signaltransmitter ten times. The experiment is made in a canteen before and after lunch and the results are summarised in Table 7.3.

Lunch	ST1	ST2	ST3
Before	-10.4	**-9.8**	-12.0
During	-13.7	**-15.5**	-13.2

Table 7.3: Table summarising the results of conducting the test in a canteen before and during lunch.

7.4.1 Conclusion

From these results, it can be concluded, that people density is a factor which cannot be neglected if high accuracy is desired. Especially the measured RSSI values for ST2 vary significantly.

Different approaches for accommodating the differences in people density have been examined and are described below:

- Construct radio maps corresponding to the times of the day where the density is known to vary significantly. This presumes that the people density in an environment is a function of time.
- Additional logic on the signaltransmitters could be used in order to determine the number of connected mobile devices in the area. This approach relies on that the number of people using the system provides a sufficient heuristic of the total number of people in the area.
- Yin et al. (2005) describes that one radio map can be used and adapted according to the current environmental conditions. They use a number of reference signaltransmitters statically placed in the environment which measure the signal strengths of the a subset of the other signaltransmitters.

  Regression analysis is used to construct a model to describe the relationship between fingerprints and reference measurements that indicate the current state of the environment. This is done for each signaltransmitter present in each fingerprint in the offline stage.

  In the online stage, the models can be used to, given a measurement of the current state of the environment, correct the radio map accordingly. Yin et al. (2005) describes that improvements in the order of 10-15% were observed in their position estimates using this particular approach.
- The authors have theorised that one potentially could use a single relationship to describe how environmental changes affect the signaltransmitters, hence making a joint regression analysis. In contrast to the above this would avoid having multiple relationships that need to be devised, hence this approach is not as complex as the one suggested by Yin et al. (2005). It is assumed that the relationship is monotone.

The approach using times of the day to choose a radio map has been assessed to not be practically feasible to positively affect fingerprinting. This is primarily based on the assessment of time not being a sufficient heuristic of people density for several reasons, such as people crowding a specific area due to a meeting that irregularly takes place or an event happening in the weekend.

The variation of using the number of users using the system to determine the people density has also been considered unrealistic, since it is in the authors opinion that the number

of people in an area cannot be determined based on the number of detectable Bluetooth devices. Even if this could be determined, a way to change the radio map accordingly also needs to be found.

The approach suggested by Yin et al. (2005) seems promising, but on the cost of adding additional information to the radio map, adding an offline processing stage, and increasing the calculations necessary for position estimation. In the offline stage, regression analysis for devising mathematical relationships need to be constructed.

If the approach suggested by the authors can be used in practice, it is beneficial due to its simplistic nature. Therefore, to verify if this holds, experiments are conducted to examine the existence of a relationship between signal strength changes to reference tags and changes to the fingerprints in the radio map. To do this, following procedure is conducted:

1. Four signaltransmitters, are set up in an area. One of them, designated the master, is chosen as the one from which reference measurements, are made to the other signaltransmitters. The master is left out in the subsequent experiment.

2. Reference measurements are made from the master to the signaltransmitters and a reference fingerprint from a predefined location is made from a mobile device to the signaltransmitters.

3. The environment is then changed. After each change, the master measures reference RSSI values of the signaltransmitters and a fingerprint is conducted at the predefined location. The changes are made using other Bluetooth devices to create interference, tinfoil to weaken the signal strengths, and people to absorb the signals. For each change this step is repeated.

4. Finally, the experiment is repeated in another environment, since, if a model exists, it must be usable in all areas.

Table 7.4 shows the results from the experiments.

Area 1							**Area 2**						
ST1	ST2	ST3	R1	R2	R3		ST1	ST2	ST3	R1	R2	R3	
-6.4	-5.5	0.1	-6.4	-5.5	-0.3	Ref	-0.1	-1.9	5.9	-7.1	-11.2	-2.7	Ref
⇑ -4.6	⇑ -4.6	⇑ 0.4	⇓ -19.7	⇓ -12.6	⇓ -3.6		⇑ 0.0	⇓ -9.9	⇓ -0.1	⇓ -14.0	⇓ -19.6	⇓ -10.6	
⇑ -4.0	⇑ -2.3	⇑ 0.3	⇓ -19.4	⇓ -12.4	⇓ -4.0		⇓ -0.4	⇓ -3.5	⇓ -0.1	⇓ -8.0	⇓ -14.0	⇓ -9.6	
⇑ -5.6	⇑ -2.7	⇑ 0.9	⇓ -19.8	⇓ -12.6	⇓-3.9		⇑ 0.0	⇓ -2.4	⇑ 6.5	⇑ -3.0	⇑ -11.0	⇓ -3.2	
⇑ -3.4	⇑ -2.9	⇑ 0.3	⇓ -20.2	⇓ -12.6	⇓ -4.4		⇓ -0.8	⇓ -4.2	⇓ 2.6	⇓ -11.4	⇓ -17.3	⇑ -2.2	
⇑ -4.8	⇑ -2.1	⇓ -0.1	⇓ -18.5	⇓-10.12	⇓-2.7		⇒ -0.1	⇓ -3.8	⇓ 4.5	⇓ -8.5	⇓ -11.3	⇑ -2.3	
							⇒ -0.1	⇓ -2.8	⇓ 3.2	⇓ -7.7	⇓ -11.5	⇓ -3.1	

Table 7.4: RSSI measurements in two areas when applying changes to the environment. The rows marked with *Ref* are the measurements made when no changes have been applied and therefore these are used as reference. The arrows indicate whether the RSSI measurements have increased or decreased in respect to the reference in the given area.

As shown in the table, the general tendency in the RSSI measurements of the reference tags is that the RSSI values decrease. However, in the fingerprint measurements, there is no general tendency. In the first area, the tendency is that the RSSI values in fingerprint increase when the RSSI values of the reference tags decrease. In the second area, the tendency is that the RSSI values in the fingerprints decrease when the RSSI values of the reference tags decrease. Based on these results, the conclusion is, that a linear, or monotone in general, relationship cannot be established between RSSI values of the fingerprint and reference tags. Thus it is concluded that this more simple approach to the problem is not applicable.

The other proposed solutions seem promising, but since focus is on examining a simpler solution, none of them are implemented in BlueCAML.

7.5 Signal Strength Fluctuations over a short Time-Frame

With regards to the offline stage, it is desirable to reduce the measurement time as this reduces the effort required for constructing the radio map. Furthermore, if fingerprints in the online stage are too time consuming, it would be more difficult to convince the user to conduct them.

To analyse the behaviour of RSSI measurements over a short time-frame, an experiment is conducted in which a mobile device collects RSSI values from a signaltransmitter over a time period of five minutes. The results are used to determine the minimum length of the time RSSI measurements need to be collected for making a sufficient accurate average. For every 250 milliseconds interval, the average of the collected RSSI values is calculated and plotted into a graph. In Figure 7.2, all RSSI values are plotted with the corresponding time frame from which they were collected.

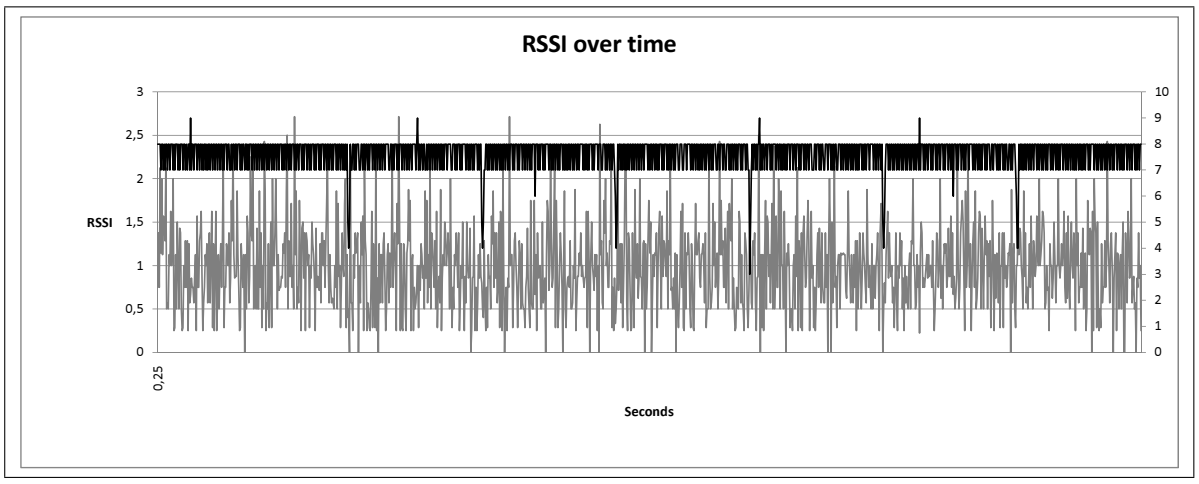


Figure 7.2: Averaged RSSI values for each 250 milliseconds time interval. The experiment was conducted over a time frame of five minutes.

7.5.1 Conclusion

In the figure, the black graph shows the number of measurements conducted in each of the intervals which shifts between seven and eight measurements. The second graph shows the RSSI values which fluctuated between 0 and 2.75 during the five minutes time-frame. The average of RSSI values over the entire time-frame was determined to be 0.98.

It is notable from the graph, that the RSSI values fluctuate a small amount over a short time-frame. In Table 7.5, the worst case and average error for different time intervals have been calculated from the entire collected data set. From the table, it can be observed that if only time intervals of 250 milliseconds are used, then the worst observed error between the calculated average and the average over five minutes is 2.4 with an average error of 0.4.

Time Frame	Worst Case Error	Average Error
250ms	2.45	0.40
500ms	1.35	0.28
750ms	0.89	0.21
1000ms	0.79	0.19
2000ms	0.39	0.13
3000ms	0.38	0.10
4000ms	0.24	0.08
5000ms	0.29	0.08

Table 7.5: Various time frames and their corresponding worst case and average error.

For the offline stage, measurement times of 2000 milliseconds which have a worst and average error of 0.4 and 0.1 are assessed to be acceptable errors. This would result in each fingerprint, with four visible signaltransmitters and four directional measurements, taking 32 seconds. The same is valid for the maintenance part.

In the online stage, a minor erroneous position calculation is acceptable as long as it can be done quickly. It has been assessed, in respect to the scenario, that the mobile device can at best collect RSSI values from four signaltransmitters at each location meaning that if a new position estimate is needed for every four seconds, minus the time used for connection establishment, which is described later, then each measurement must at most use approximately 500 milliseconds for the actual position estimate calculation.

7.6 Density Of Fingerprints

Before conducting the offline stage, one must decide upon the density of the fingerprints, that is, the distance between consecutive fingerprints in the radio map. To do this, one should take into account the purpose of the system. If the purpose is to provide relatively accurate

tracking of a movable object, then the density should be high to provide a good estimate of the position of the object. In contrast, if only positioning at room-level is required, the fingerprints can be relatively sparse.

7.6.1 Conclusion

As described in Chapter 3, the purpose of BlueCAML is to enable the user to follow her movement in an indoor environment. Thus, it is desirable to have as high accuracy as possible. The effort required in creating the radio map must also be taken into account. If a lower accuracy of the system is enough, then having a lower density of fingerprints is preferable since it requires less effort and maintenance when working with the radio maps.

In previous research (Sw701b, 2009) it has been shown that two to three meters movement is required in order to have an observable change in RSSI value. Thus, is desirable to place fingerprints with three meters interval, such that they are distinguishable. This density has also been used by other indoor positioning systems (Hansen and Thomsen, 2009).

7.7 Time Measurements of the Fingerprint Process

In order to create a fingerprint and estimate the position of a mobile device, a number of RSSI measurements needs to be collected in a limited amount of time. A goal of BlueCAML is to be able to update the position estimate at least every four seconds, as stated in the requirements. Therefore, it must be examined if the measurements can be made within this period of time.

The phases involved in position estimation are identified as the following:

- Discovering detectable signaltransmitters in the area.
- Establishing connections between the mobile devices and the signaltransmitters.
- Measuring RSSI values from the signaltransmitters.
- Calculating the position based on the measurements.

Experiments are conducted to determine the required time for these phases. In the first phase, a discovery search refers to the mobile device searching for visible signaltransmitters by hopping between different frequencies. Since the Bluetooth devices are not synchronised, they do not know the sequence of frequencies the signaltransmitters are hopping between. Consequently, a relatively high amount of time is needed in order to discover all devices. According to Woodings et al. (2002), approximately ten seconds are necessary if following the procedure defined by the Bluetooth specification (SIG, 2007). It is possible to limit the time used on searching, but this would potentially only return a subset of the visible devices. The theoretical discovery time has been confirmed by experiments conducted by the authors where 60 discovery searches in average took 10.2 seconds.

To determine the required time for the second phase, a single mobile device is continuously connecting and disconnecting from a signaltransmitter while registering the connection times. The experiment is conducted until 500 connections have been established. This number is considered appropriate for providing the basis for calculating the average connection time. The experiment shows that a connection in average takes 3.7 seconds. To make sure that the connection time in itself does not exceed the requirement in an actual use of the system, Bluetooth load is put on the signaltransmitter, simulating multiple users connecting to it. The result of doing this shows that the average connection time does not exceed 3.7 seconds.

Experiments to examine the time used for the third phase, has already been conducted and showed that RSSI values should be measured for 500 milliseconds to each signaltransmitter.

Finally, experiments are conducted on the mobile device to determine the average time needed for the calculations related to position estimations.

Table 7.6 summarises the findings in this experiment.

Discover	Connection	Measure RSSI	Calculations
$10s$	$3.7s$	$500ms$	50ms

Table 7.6: Overview of time usage of different operations related to position estimation.

7.7.1 Conclusion

As described, the discovery and connection phases use a considerable amount of time.

The connection time is 3.7 seconds the position estimate is above the four seconds requirement, and measuring RSSI values from four signaltransmitter and conducting one position estimate, this requires 5750 milliseconds.

In order to reduce the time needed for position estimation, a strategy is needed. Assuming that it is possible to compute which signaltransmitter is likely to be detectable given the previous estimated position, it is possible to maintain a list of connected signaltransmitters, by connecting to one new visible signaltransmitter for each position estimate. This approach seems usable given that the available signaltransmitters does not change substantially over short distances. If the system is unable to use the current estimated location to determine which signaltransmitters it should connect to, then it could fall back to use the discovery search. However, due to the required amount of time for conducting this operation it should in general be avoided. Also, the discovery can be used as an initialisation phase where signaltransmitters are detected to ground the basis for further processing.

As described, the time required for the discovery phase can be reduced by only discovering a subset of the detectable signaltransmitters. However, in the authors' opinion, a decrease from ten seconds to e.g. six seconds does not matter since the time required for the task still exceeds the requirement of four seconds. Therefore, if the time requirement is to be exceeded, it is preferable to ensure that all detectable signaltransmitters are discovered.

8

Test and Verification Methods

This chapter gives an overview of testing fundamentals which are used throughout the project in order to ensure conformance to the requirements and reduce faults. Also, the chapter complies with the learning goal concerning test and verification.

There exist different definitions of the terms test and verification. In this chapter, the following definitions are used conforming to the *Test and Verification* course the authors have attended during the work on this project (Mathur, 2008):

Testing Testing is aimed at *detecting* faults in a program.

Verification Verification is aimed at *proving the correctness* of a program according to a specification.

Using testing, one can execute an application or parts of it with different inputs in order to find possible faults caused by these. If no faults are revealed, then one cannot assure that there are no faults, but only that the program functions with the tested input.

With verification, one proves that the program is correct with respect to the specification. This requires that the specification is correct and the methods used to prove this do not themselves contain faults.

Both of these approaches are described in further detail in the following.

8.1 Testing

The following describes different ways to categories a test and how tests can be constructed. Mathur (2008) and Sommerville (1999) are used as reference.

Different sources for constructing tests can be derived, how tests are applied to different phases of the development, how tests can be generated, and, finally, how test coverage can be determined, are described.

8.1.1 Test Sources

Tests can be derived from different sources such as requirements and the source code itself. The different test sources allow the tests to focus on different aspects of the program.

Informal Requirements Informal requirements, such as requirements expressed in natural language, are often used when generating tests. The requirements specify functions, their inputs, their outputs, and expected behaviour, which can be transformed into tests by a developer. A problem with informal requirements is that they can easily be expressed ambiguously and tests derived from these can thus easily contain faults in regards to the desired behaviour.

Formal Requirements Formal requirements are expressed using mathematical expressions, models, or some other formal representation. In relation to informal requirements these are easier to express without ambiguity. Furthermore, since they can be expressed without ambiguity they can be used for automatically generating tests.

Source Code If the source code is available, tests can be constructed by evaluating the actual implementation and the internal logic. Access to the source code also allows for evaluating the test coverage of the code in order to measure how much is tested and ensure that the most critical parts of the source code are covered through tests.

The different sources for test generation are suitable for different approaches to testing. The formal requirements approach requires the presence of a precise specification, defining all input, output, and behaviour in an unambiguous format. This is not used in this project, since the requirements are expected to change, it is estimated that maintaining a formal specification would be too time consuming.

Thus, a combination of the informal requirements and source code are used in order to generate the tests. This involves creating tests from the requirements and internal logic.

8.1.2 Tests Applied to Different Development Phases

Different types of testing can be conducted in different phases of the development. These different types of tests are as follows:

Unit Testing Unit testing is conducted in the coding phase and focuses on individual components, such as a class or an individual function. These are tested in isolation from the surrounding system.

Integration Testing Integration testing is conducted in the integration phase, in which the developer takes a newly developed component and integrates it into other existing components. This type of testing thus focuses on the interoperability between multiple components.

System Testing System testing is conducted on the entire system. Thus, this type of testing involves all components of the system and checks whether the system fulfils its requirements.

Regression Testing Regression testing is conducted when changes are made to the system, and it must be verified that the changes do not break any existing requirements. Thus tests selected for regression testing consist of those directly related to the changed code and parts of the system which could have been affected by the change.

Acceptance Testing Acceptance testing is used to approve a release of a program. This is done by constructing tests which exercise the requirements, in an environment which is as close as possible to the anticipated operation environment. This type of test can then be used to decide whether the program is ready for release or not.

Beta-Testing Beta-testing consists of end-users getting access to the system before being finalised, such that they can test it through normal use. The main difference between this type of testing and usability testing is the way it is conducted and when they take place in the development phases. Usability tests can be used throughout the phases whereas beta-testing is conducted in the end of the development. Furthermore, usability testing is directly focused on how the users interact with and use the application, while beta-testing is conducted by the users themselves and can uncover problems ranging from usability problems to functional problems.

In this project, all of the above testing methods except beta-testing are used. Beta-testing is omitted since testing of the system is conducted by the authors themselves, and possible external input from others is covered using usability tests.

Unit testing and integration testing are conducted throughout the development using automatic testing tools. This ensures that these can be conducted quickly and often, which allows for easy regression testing when changes are made to the system.

System testing is conducted manually. This is because of the structure of BlueCAML which consists of multiple applications running on different machines. This adds some difficulty in automatically testing BlueCAML as a single entity since actions need to be initiated on one machine which results in changes on another. Acceptance tests are conducted as one of the final tasks of the project in order to verify that the final system meets the requirements.

8.1.3 Test Generation

Different approaches exist for creating the tests, directed at source code, in an efficient manner. The following describes four different approaches to this:

Equivalence Partitioning Testing using equivalence partitioning involves analysing the input domain of all inputs to the application, and dividing it into input equivalence classes. All inputs in an equivalence class should result in the same behaviour of the application, and thus only one input from each equivalence class needs to be tested.

As an example, if a function accepts all positive integers, one can create two equivalence classes: one with all positive integers, and one with all negative integers, and test the behaviour of the application from a sample from each of these classes.

Boundary-Value Analysis A common place for failures to occur is in the boundaries of equivalence classes. E.g. if an equivalence class consists of the positive numbers below 10 and another consists of the positive numbers greater than or equal to 10, then there exist a boundary at 10. This boundary is then tested by examining the boundary values 9, 10 and 11.

Since boundary analysis uses equivalence classes, some overlap exists between tests constructed with these methods.

Cause Effect Graphing Cause effect graphs builds on the notion of causes, which are input to the application, and effects, which are output from the application. The different causes are then connected with the effects in a graph, together with edges indicating relations such as dependencies and exclusion. The resulting graph can then be used to generate tests, based on the required inputs used to cause the different effects.

Finite-State Models (FSM) The specification of an application can be used to create an FSM which in turn can be used to auto-generate tests. These tests can then check if the application conforms to the behaviour described in the model, ensuring its conformance to the specification.

From the above, only two of the test generation methods are used in this project: equivalence partitioning and boundary-value analysis. These are selected based on their simplicity.

The remaining methods require representing the requirements in some intermediate form which then is used to auto-generate tests. This is considered too comprehensive for this project.

8.1.4 Test Adequacy

Test adequacy concerns determining whether the tests are sufficient for their purpose. In order to do this, a number of methods can be used. Three of these, which have been described by Mathur (2008), are listed in the following.

Control Flow Control flow expresses the test coverage of the application in terms of either statements, blocks of statements, conditions, or decisions which have been evaluated under test. This method thus expresses how much of the application is examined when executing the tests and can indicate if there exists parts of the application which are uncovered.

Data Flow Data flow adequacy is a measure of the test coverage of changes to and usage of data in the application. In contrast to the control flow adequacy method, data flow

adequacy measures how many of all possible paths, in respect to data, are examined as part of the tests.

Mutation Test adequacy assessment through mutation is considered a white-box technique in which the original source code is changed slightly. The changed program is denoted as the mutant. The incentive to conduct mutation can be explained through a simple example. Consider the expression $boundary < max\_val$ which has been tested. In this case one might ask whether this is the correct expression or whether $boundary < max\_val + 1$ is the correct one. To prove the alternate solution correct or wrong, one can simply mutate the original source code, run the test, and verify whether or not the mutant behaves differently.

In respect to this project, it is not important which method is used, since the main purpose of this is to ensure that components are not by accident neglected when testing.

From the described methods, the used IDE can be extended with NCover (NCover, 2010) which is a program that supports control flow adequacy. This method is therefore used.

8.2 Verification

In the *Test and Verification* course, model checking has been introduced for verification. In model checking, a formal model is constructed, e.g. by using finite-state automata. The process of constructing the model can be done by examining the specification of the software. Model checking allows for checking whether or not the software upholds certain properties that are essential for proving its correctness. The result of checking properties can be one of three different outcomes: satisfied, not satisfied, and unable to determine. The last case may be present in situations where the model checker is incapable of properly terminate due to an upper bound of iterations being exceeded or other such limitations.(Mathur, 2008)

When conducting model checking, with verification purposes, one must in most cases try to limit the verification space of the software due to the combinatorial explosion of the state space which results in a time consuming process or in the worst case a problem being practically impossible to verify due to its size.(Mathur, 2008)

To verify models, model checkers such as UPPAAL (Bengtsson et al., 1996), TAPAs (Calzolai et al., 2008) and PRISM (Hinton et al., 2006) can be used. Since lectures have been given on UPPAAL, and a more in-depth analysis of the possible model checkers is out of scope for this project, UPPAAL is chosen.

One should be aware, that by using a tool like UPPAAL to verify properties of a model, gives rise to some concerns. These include, that if the tool verifies a certain property, the reliability of the answer relies on a correct implementation of the tool. Also, in that context, the tool cannot determine whether the model is in accordance to the specification. Finally, verifying properties of a model only describe the behaviour of the model and not the actual implementation. Therefore, if for instance the model does not yield a deadlock, the implemen-

tation must be exactly equal to the model before it can be inferred that the implementation does not yield a deadlock neither.

In the following, UPPAAL is introduced.

Introduction to UPPAAL

UPPAAL uses timed automata to model a system for which certain properties can be verified. As an example, the property of deadlock can be verified, by a thorough search that covers all dynamic behaviours of the system.(David and Amnell, 2002)

UPPAAL comprises a model checker engine which can be used either directly or through a GUI. This makes it possible to use UPPAAL both on normal desktop computers for designing the models and using more powerful servers to do the actual model checking.

The following description of the UPPAAL notation builds upon David and Amnell (2002), Bengtsson et al. (1996), and University and University (2006).

A UPPAAL model consists of a number of timed automata, each representing parts of the system. These are called templates and consist of locations, edges, and a number of attributes on these.

For the locations, the following attributes are used in this project:

Name The locations can have names identifying them. These can be referred to when checking certain properties.

Initial State Each template must have exactly one initial state. This state is marked with a double circle.

Invariants The invariants can be used as progress conditions, that is, if a location is labelled with an invariant, the system must only be in that location as long as the invariant is true. For instance, if a location is annotated with the invariant $time <= 5$ where time is a clock variable, then the system is not allowed to stay in that location for more than five time units. Clock variables are used to keep track of time and are automatically incremented whenever the model changes state.

Committed Locations If a location is marked committed, represented by a C, it essentially means that, when the location is entered, time is frozen and the next transition must involve at least one of the outgoing edges of the committed locations. Hence atomic actions can be simulated.

The edges can be annotated with the following:

Selections Is used to non-deterministically bind an identifier to a value given a scalar set or a bounded integer. Thus, selections can be used to select an element of a set in order to represent a specific process from a list of processes.

Guards If an edge is annotated with a guard, it means that the automaton is only able to fire the particular edge if the guard expression evaluates to true. As an example, if one wants to fire an edge if and only if the clock variable *time* is above five time units, one would place the $time > 5$ guard on the edge.

Synchronisation Two running templates, called processes, can synchronise over channels using complementary actions. E.g. when defining a channel called c, using c! in one template represents a sent signal through the channel and c? represents receiving the signal by another process. When two processes synchronise, both edges are fired at the same time, meaning that the two involved processes change location.

The previously described synchronisation is called a binary synchronisation. Another variant is making broadcast channels. This variant of synchronisation allows a sender, c!, to synchronise with multiple receivers, c?.

Updates The update expression is executed when the edge is fired. This can be used to make variable assignments.

As described previously, the purpose of a model checker is to verify the model with respect to a requirement specification. To do this, the specification must be expressed in a formal language of which UPPAAL makes use of a query language consisting of path formulae and state formulae. As the name implies, path formulae expresses paths of the model and state formulae describe the state.

State formulae are expressions in UPPAAL that can be evaluated like the boolean expression $var == 5$. State formulae also include testing whether a particular process is in a particular location which is expressed as $Proc.loc$ where $Proc$ denotes the process and Loc the location. In addition, UPPAAL also makes available a special keyword, *deadlock*, which is the state formula evaluating to true for all deadlock states. This is a convenient state formula to use in the verification of the model as it enables testing whether or not the model can result in a deadlock state.

The path formulae can be classified into three different property classes, namely: reachability, safety, and liveness. Following is a brief description of each of these.

Reachability Properties of this class ask whether a given state formula, φ, possibly can be satisfied by any reachable state. In UPPAAL this type of property is expressed as $E\Diamond\varphi$.

Safety These properties are used to describe that some state formula will possibly or never occur. For instance, it may be convenient to verify that a deadlock never occurs in a model. Provided that φ denotes a state formula, $A\Box\varphi$ says that in all reachable states, φ should be true. On the other hand, $E\Box\varphi$ says that there should exist a maximal path such that the state formula, φ, is always true.

Liveness Properties of this type are used to describe that something will eventually happen. In UPPAAL, these properties are described using the path formula $A\Diamond\varphi$ which describes that the state formula, φ, is eventually satisfied. However, a more convenient property is the response written as $\varphi \rightsquigarrow \psi$ which means that whenever φ is satisfied, then eventually, ψ will be satisfied.

Part II

Architecture

9 Technical Platform

The purpose of this chapter is to highlight the technical aspects of the constituents of BlueCAML. Specifically, this chapter emphasises on determining the appropriate implementation languages and prepares for which technologies will be part of the system.

With relation to the mobile application, various essential choices need to be made. Among others these comprises determining which language and platform provide the necessary functionality for enabling that positioning can be performed. Also, it is important that the correct map format is decided upon.

The back-end is only supposed to store the radio map and expose an API for manipulating and retrieving it through services. In this regard, the appropriate programming language is chosen.

Finally, an implementation language for the logic residing on the signaltransmitter is decided upon, and in addition the appropriate network topology is found.

9.1 Back-end

On a previous project (Sw701b, 2009), knowledge was acquainted with Ruby on Rails which proved to be capable of quickly prototyping applications especially in regards to simple data storage and retrieval through web services. For this project, Ruby on Rails 2.3 along with its default web server WebBrick are used. The back-end is installed on Windows XP. In a deployment situation, the back-end can easily be migrated to other platforms supporting Ruby on Rails.

Ruby on Rails uses RESTful interfaces by default. This means that the web services comply with the architectural principles of REpresentational State Transfer (REST) defined below (Rodriguez, 2008):

- The GET, POST, PUT and DELETE HTTP methods should be used. GET is used when retrieving a resource, POST when creating a resource, PUT when updating a resource, and DELETE when deleting a resource.
- The web service must be stateless, meaning that session data must not be stored.

- The URIs used in REST web services should be easy to guess the meaning of. Therefore, directory structure-like URIs are encouraged.
- The response data must be returned in the body of the HTTP response and is typically either represented in; JSON, XML or XHTML.

9.2 Mobile Device

In respect to the mobile application, a programming language, data storage type, and map format must be defined. This is done in the following sections.

9.2.1 Programming Language

Since some programming languages are platform dependent, the choice of programming language depends on the specific platform. The available hardware with a corresponding platform are listed below:

- iPhone 3G - iPhone OS
- HTC Diamond Touch - Windows Mobile 6.0
- Nokia E66 - Symbian OS v9.2

Since the development of BlueCAML is primarily focused on the applicability of Bluetooth for indoor positioning, the authors have not found it crucial to find alternatives to the available hardware. If it had been crucial, Google's Android would be interesting to examine further, based on its lately, almost explosive, growth in market share.(Canalys, 2010)

The mobile devices open for the usage of Objective-C using Cocoa, Java using Java Micro Edition (Java ME), languages supported by the .NET Compact Framework (.NET CF) including C#, Visual Basic and Visual C++, and Symbian C++ using S60 3rd Edition.

Programs written in Objective-C can be compiled for iPhone and is a small extension to the ANSI C programming language that supports the object oriented programming paradigm.(Apple, 2009)

Programs written in Java can be compiled for the Java Virtual Machine (JVM) and run on the HTC and Nokia mobile devices, hence opens for platform independent development. The Java ME platform is a subset of Java SE and its purpose is to make it possible to develop Java applications for mobile phones, embedded devices etc. with limited resources (OracleSDN, 2010).

C# compiled code can be run on the HTC Diamond Touch using .NET CF which is a downscaled version of the .NET Framework where components not relevant for mobile development has been excluded (Fitzek, 2009, c. 8).

Symbian C++ is used for developing software for the Symbian platform. It is a specialised subset of C++ and can result in a steeper learning curve (Foundation, 2010).

In order to determine which platform and programming language would suit the requirements best, a brief analysis is conducted in which their features and possibilities are examined. The first question to ask in this analysis is whether or not the platform and corresponding programming language provide a means of collecting RSSI values from the Bluetooth device. This question is alpha and omega because, as outlined previously, the mobile device is responsible for collecting the RSSI values. In determining whether or not this is possible, the documentation of the individual platforms and programming languages are consulted and the conclusion is that Symbian, iPhone and Java ME do not implement such features. However, Windows Mobile with the .NET CF does and is therefore used.

Even though the project aims at showing the applicability of Bluetooth for an indoor positioning system, Java ME would have been desirable because of its *write once, run anywhere* design philosophy. This lets the program run on virtually any device equipped with a JVM provided that they implement the same configuration, hence making it cross-platform. Because of this, a system developed using Java ME undoubtedly targets a wider market segment.

According to an analysis conducted by Canalys (2010), the leading smart phone operating system vendors as of 2009 are, in decreasing shipments, Symbian, RIM, Apple, Microsoft and Google. As of this writing, all of these except Apple's iPhone and Google's Android natively employ or give by default the possibility of employing a JVM, thus, they are capable of executing the same program, provided that they implement the same configuration. Note that there exists solutions for converting Java ME applications to iPhone applications.(DEVELOPMENT, 2008; Symbian, 2010; Sun, 2005; Burnette, 2007; Mysaifu, 2010a,b)

The .NET CF provides the opportunity of using various languages and even let them intermix seamlessly. The choice of appropriate language is described below.

.NET Compact Framework

As previously described, the .NET CF is, in respect to the .NET Framework, a smaller, more client oriented framework (Yao and Durant, 2010). (Fitzek, 2009, c. 8) describes that the size of .NET CF is around eight percent of the full .NET Framework, meaning that the framework can fit into devices with limited memory resources. It consists of the base class libraries and includes functionality targeting the mobile platforms such as special forms.

The Common Language Runtime (CLR) is placed on top of the operating system and its purpose is to execute the .NET byte code on the platform. To make this process more efficient and economising the power consumption, the CLR has been developed from scratch for the .NET CF.(Barnes, 2010)

Using the CLR to execute the application, opens for development in different programming languages, such as C#, Visual Basic, and Visual C++. Using C# has been assessed to be

an advantage because of prior experience. Also, it is not considered beneficial to use two programming languages for development since there is no advantage in doing so for developing the mobile application. It is therefore developed using C# and the .NET CF 3.5. Utilising an intermix of programming languages could be desirable in situations where one wants to accommodate the various preferences of developers.

9.2.2 Storage of Data on the Mobile Device

The mobile application needs to store radio maps. Some considerations are needed when choosing how to store this data on the mobile device. The data is to be loaded into memory on initialisation, so the speed of retrieving the data from the storage is less important.

For storing the data, two alternatives have been identified, namely a database such as a spatial database or using XML. It has been identified that using a database would not introduce advantages of significance. In contrast, however, depending on the database, it would introduce another dependency to an external library whereas .NET contains built-in functionality for manipulating XML entities etc. Also, the fact that potential geometrical operations can be made by mobile application itself, minimises the advantages of using a spatial database considerably. Therefore, the radio maps are stored in XML files in BlueCAML.

9.2.3 Choice of Map Format

A map of the current environment is necessary in order to show the estimated position visually to the user. In order to render this map, two possible solutions are evaluated; a pre-rendered raster image and rendering the environment using IFC models.

Using a raster image would require that an image is made for each supported environment, either manually or by using pre-existing technical documents which are corrected. A drawback is the inability to change the visual representation of the map without re-rendering the images, such as if the colour and thickness of the walls need to be adjusted.

In contrast, this is resolved using IFC models. These models contain detailed information about buildings such as walls, windows, doors, and even electrical wiring. This allows for rendering the environment based on this information and changing the renderer if necessary. All buildings built or renovated by the Danish government with a budget above 20 million are required to have an IFC model (DetDigitaleByggeri, 2010). Using the IFC model, one can render the environment both in 2D or 3D in contrast to the 2D format of a raster image which shows the flexibility of using IFC in relation to raster images.

Due to the high level of detail provided in the IFC models, they tend to become quite large. However, it has been successfully used in mobile devices in the research project conducted by Ferial Shayeganfar and Tjoa (2008). It has been assessed, that the full IFC model is not the best solution in BlueCAML. This is both due to the unnecessary level of detail and the requirement of conducting position estimation on the mobile device which potentially require high computational effort. Fortunately, a variant of the IFC model has been developed, namely

the Portable IFC (PIFC) (Šikšnys, 2010). The PIFC format extracts relevant information from the IFC model and defines the position of walls, defines the names of different areas such as offices, and annotates the positions of relevant elements such as Wi-Fi access points. All other data kept in the IFC model such as plumbing and wiring are not taken into account. This means that the file size of PIFC is significantly reduced in respect to the original IFC file from which it was derived. As an example, an IFC file for the building for Department of Computer Science at Aalborg University was reduced from 9.5 MB to a PIFC file of size 141 KB. The PIFC model is represented in XML format, which must be read by the renderer in order to extract the coordinates for the different parts of the environment before rendering the map.

Another variant is the CityGML model (CityGML, 2010) which similarly is a less detailed version of the IFC model and provides the opportunity of specifying the detail-level of the individual objects. Finally, CityGML aims at providing detailed outdoor objects such as streets or vegetation which may be necessary if one wants to model the objects of a city (Stephan Mäs and Wang, 2008). Customisation with the PIFC model is possible such that signaltransmitters, for instance, are represented also. CityGML could potentially be used, however, it is more a format aiming at describing objects that may exist in an outdoor environment which is not necessary in the case of BlueCAML.

Between PIFC and CityGML, it has been determined that PIFC is the proper choice due to its detail level being in accordance with the purpose of BlueCAML, that is, walls, doors and other elementary objects for defining the structure of a building are representable.

It has been assessed that using a raster image or the PIFC model are equal in regards to required effort in their implementation. From the authors experience, displaying a raster image in a GUI and drawing additional information, such as the current position on top of the image, is fairly simple (Jensen et al., 2007). Rendering the PIFC model would require more effort, but an implementation already exists for rendering the model in 2D on mobile phones. By reusing this implementation, the effort would be greatly reduced.

One added concern is the increased requirements for rendering the map on the mobile device from the PIFC format. However, experience from the existing implementation of the 2D renderer shows that a modern mobile phone is capable of rendering it fluently. Due to this reason, the PIFC model is chosen as map format on the mobile application.

9.3 Signaltransmitter

Regarding the signaltransmitters, the following sections describe the hardware specification, programming language, and network topology of them.

9.3.1 Hardware

The infrastructure of BlueCAML is similar to the one used in Easy Clocking (Sw701b, 2009), meaning that the hardware remain the same. The specification of the signaltransmitters is shown in Table 9.1 (DD-WRT, 2009).

Device	Asus WL-500gP V2
Price	700 Dk/kr
Firmware	OpenWRT Kamikaze 8.09.1
Linux Kernel	2.4.35
Platform	Broadcom 5354 Chipset
CPU	MIPS32 CPU running at 240 MHz
Flash memory	8 MB NAND
System memory	32 MB 16-bit DDR SDRAM
USB ports	2 x USB 2.0
Wireless radio	Broadcom 802.11b/g
Network switch	4 × 10/100 Mbit LAN and 1 × 10/100 Mbit WAN

Table 9.1: Hardware specifications for the signaltransmitters (DD-WRT, 2009).

Other hardware than the listed could be used as long as the device can support making the Bluetooth adapter discoverable to the environment. Depending on whether a smart or dumb signaltransmitter is set for deployment, the device requires network capabilities in case a smart signaltransmitter is desired. Thus, the requirements for them are fairly low.

As shown in the table, the signaltransmitters use the OpenWRT firmware which is an open source Linux based firmware for embedded devices. This means that it is applicable by a number of devices, hence the signaltransmitter application should be executable on any of these as long as they have this firmware.(OpenWrt, 2010)

The signaltransmitters are based on the MIPS architecture meaning that a cross-compilation toolchain is required for generating the executable when developing on an x86 based architecture. OpenWRT makes available such a cross-compilation toolchain.(OpenWrt, 2010)

9.3.2 Programming Language

For programming embedded devices, a variety of programming languages are possible. At the previous project, C was used for programming the WL500gP router. The experiences with this language was generally positive, but as the size of the application grew, it gradually tended to be more difficult to maintain an overview of where different functionality was located in the source. During the construction of that application, it was also frequently necessary to construct elementary data structures with accompanying helper functions for manipulating and accessing them. Due to these experiences, together with the desire of applying a new

and so far untried programming language for embedded software development, it has been decided to try a new one.

In this project, C++ has been decided as implementation language for the application residing on the signaltransmitter. The primary argument for this choice is due to language features and extensions over C that makes object oriented programming convenient in C++. The main features of the object oriented programming paradigm such as promoting inheritance, encapsulation, abstraction and polymorphism are some of the language features which could accommodate some of the experiences acquired with C. Additionally, C++ supports the full C and C++ standard libraries and makes available the Standard Template Library (STL) which contains a number of useful and elementary templates through the support for generic programming. Among others, the STL contains common data structures such as lists, vectors, and hash maps which are conveniently applied without much effort as in contrast to the requirement of manually implementing them in C.

Many programming languages supporting the object oriented programming paradigm contain aforementioned features. However, as will become apparent later, the signaltransmitter needs to communicate with a web server using the HTTP protocol for being compliant with the RESTful interface. In the previous project, similar functionality was required and much experience was acquired in using the HTTP transfer library libCURL for C. Also, experience was acquired in in-memory XML parsing and XPath evaluations using the libxml2 XML library which also will be an essential part of the signaltransmitter. Both libraries are also compliant with C++ due to their usage of language linkage to C++. Essentially, this means that acquired knowledge in these two libraries can be transferred directly to an implementation in C++. This is the primary argument for choosing C++ instead of for instance Java which is also possible due to available virtual machines for the MIPS architecture. The incentive of using Java could have been due its platform independent nature thereby promoting portability through the *write once, run anywhere* principle. However, for this project, portability is not an issue that will be taken into account, and should the necessity arise due to different hardware for instance, GNU cross-compilation toolchains exist for the majority of different architectures.

9.3.3 Network Topology

The network topology used in the previous project was based on a mesh network which was realised through the utilisation of the Optimised Link State Routing (OLSR) protocol. The primary incentive of basing the topology on this, is that it enables a relatively effortless installation of new signaltransmitters in case new areas of the particular building become of interest. The installation comprises physically setting up the signaltransmitter and make it part of the mesh network. Afterwards, the signaltransmitter is capable of wirelessly communicating with the back-end by routing data through other nodes of the network. However, it was discovered that the implementation of the OLSR protocol for OpenWRT was very unstable and practically insufficient for enabling a mesh network to successfully be constructed. Since no alternative solution has been found for this project, the network topology is based

on the star topology meaning that the nodes are connected to a central router of which may be part of another star topology and so on such that data from a signaltransmitter is routed through these until reaching the destination, the back-end, eventually. However, in an actual deployment setting, the mesh network topology could be set up statically by defining the appropriate routes in each of the signaltransmitters.

10 System Architecture

The purpose of this chapter is to describe the system architecture of BlueCAML, which helps ease the understanding of the general structure and dependencies of the system (Munk-Madsen et al., 2000). During the agile development process, the architecture has been iteratively updated and could therefore be used as a reference during development, since it forms the basis for the design.

10.1 Architecture

The following introduces the architecture, shown in Figure 10.1, which is comprised of the three primary packages: the mobile application, the back-end application, and the signaltransmitter application. The responsibilities of each of these and their sub-packages are described throughout the chapter.

Three architectural patterns are used, namely the Model View Controller (MVC) pattern for the back-end application, the Model View Presenter (MVP) pattern for the mobile application, and the layered architecture which is used both in the signaltransmitter application and the mobile application. These patterns are described in the sections corresponding to the specific packages in which they are applied.

10.2 Back-end Application

The purpose of the back-end application is to store the radio map at a central location, such that it can be distributed to the mobile devices as needed. It is also responsible for receiving corrections to the radio map and modifying it accordingly. These tasks are accomplished by providing RESTful web services to the signaltransmitters, which in turn handles requests from the mobile devices.

The back-end application follows the MVC (Fowler, 2006b) architectural pattern which is the conventional way to structure Ruby on Rails back-end applications (RailsGuides, 2009). This pattern separates concerns in the application, such as the presentation layer from the

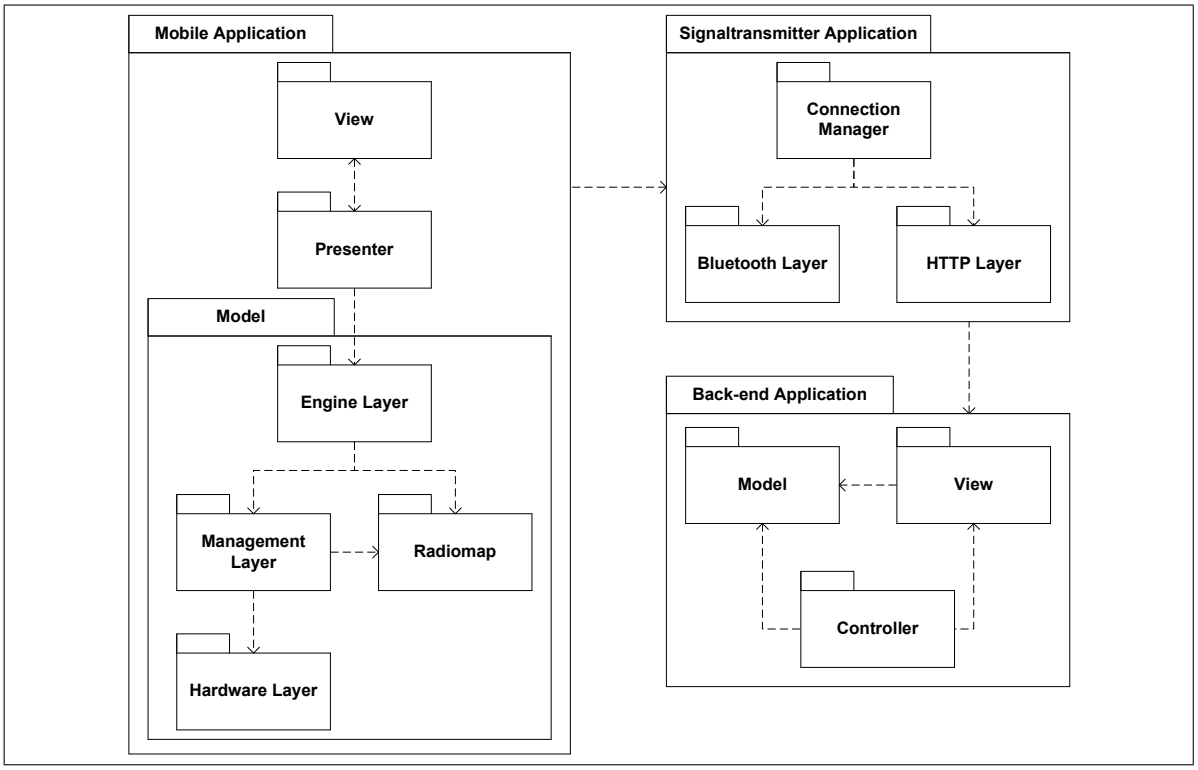


Figure 10.1: UML package diagram, depicting an overview of the three main packages comprising BlueCAML: the mobile application, the back-end application, and the signaltransmitter application. Dependencies are depicted by arrows.

model and input handling from view generation. The following describes the model, view, and the controller in the back-end application.

Model The Model contains all business logic and stores all information related to the application. This consists of the radio map and logic used to manipulate it.

View The View is responsible for rendering pages to the user based on data provided by the controller. Thus, there exist views for each of the provided RESTful web services for retrieving radio maps and uploading corrections to the radio map.

Controller The controller is responsible for handling user input which is limited to the calls to the RESTful web services from the signaltransmitters. When called, it must first manipulate the models before handing the necessary information to a View which renders a response to the calling signaltransmitter.

10.3 Signaltransmitter Application

The signaltransmitter application consists of three packages, namely: the Bluetooth Layer, HTTP Layer, and Connection Manager. The architectural pattern constituting these packages is based on the layered architecture in which the Bluetooth Layer and the HTTP Layer encapsulate relatively low level functionality providing a higher level of abstraction through interfaces to the upper layer; the Connection Manager. Besides providing a higher level of abstraction to upper layers, the layered architecture is also convenient in situations where some functionality in a lower layer needs to be changed. This is due to the layered architecture emphasising on enhancing cohesion and lower coupling.

In the following, the three packages are described individually.

Bluetooth Layer The Bluetooth Layer comprises all functionality concerning Bluetooth in the signaltransmitter application. Specifically, this layer interacts with the local Bluetooth adapter in order to measure RSSI values and handles communication with the mobile devices through Bluetooth. In case the Bluetooth stack of the signaltransmitter is substituted by another implementation than originally used, the implementation of the Connection Manager remains unchanged as long as the new implementation of the Bluetooth Layer complies with the same interface.

HTTP Layer This layer provides functionality used for communicating with the back-end through the RESTful web services provided by it.

Connection Manager The Connection Manager aggregates the functionality of the two lower layers and is therefore responsible for processing incoming requests from the mobile application and take actions accordingly.

10.4 Mobile Application

The mobile application package, follows the idea behind the MVP architectural design pattern. More specifically it is based on the Passive View variant of MVP (Corporation, 2008; Fowler, 2006a). The general concept of MVP is to separate the concerns of the user interface and the model, consisting of business logic and application state, by using a presenter. The main difference between MVP and MVC is that in MVP, only the presenter interacts with the model, whereas in MVC both the view and controller is allowed to do this.

The reason why the mobile application only to some extent follow the idea behind the pattern instead of actually applying it, is that different sources disagree on how it should be applied (Corporation, 2008; Fowler, 2006a; Cerrada, 2008). Some describe that a view should be present for each resource in the model, while others do not explicitly state this. In BlueCAML, there is one view, since there is no need to present the user with actual resources. In other applications, e.g. an order management system, one could imagine having different views for orders, customers, etc.

The Passive View design pattern has a number of side-effects besides providing a structure for controlling user interaction. By decoupling the view and the presenter, it is possible to replace the view depending on where it is used. E.g. one could imagine an application having a view for desktop usage and one for mobile device usage. Also, if BlueCAML is extended with functionality for e.g. not showing maps but only context-aware information to the users, this could be implemented as a separate view.

Another side-effect is increased testability. The Presenter contains the logic related to user interaction, which can be called directly and tested. The user interface only contains the minimum amount of logic needed to control the user interface, thereby reducing the amount of logic not tested if tests are omitted for it. Using the pattern therefore complies with the testability quality factor.

The different packages in the mobile application architecture are described below.

Model The Model contains all business logic and stores the application data and state. It is further divided into the following layers, whom are based on layered architecture.

- Engine Layer The Engine Layer is responsible for the high level logic of the application, namely: estimating the position of the mobile device and maintaining the radio map.
- Management Layer The Management Layer consists of business logic used by the Engine Layer in order to e.g. collect fingerprints, and maintain connections to signal-transmitters.
- Radio map The Radio Map package contains the radio map information and provides functions used to manipulate and compare elements in it. This package is shared between the Engine Layer and the Management Layer.
- Hardware Layer The Hardware Layer encapsulates all Bluetooth related functionality which are dependent on the hardware on which the mobile application runs. The purpose of the layer is to be able to substitute the layer according to the underlying hardware if necessary.

View The View is responsible for the GUI and initial handling of user interaction. It also contains the map which is used to show the user's current location.

As selected in Section 9.2.3, the map is rendered from a PIFC model. In order to read and render this model to the screen, a pre-existing package is used. Throughout the report, this package is referred to as the PIFC Renderer. Since it is not the focus of this project to create a PIFC Renderer, it has been decided to reuse this component.

Presenter The Presenter is responsible for handling the user interaction, by calling the appropriate methods in the Engine Layer in order to make position estimates and maintain the radio map. Furthermore, it is capable of calling methods in the View to show the result to the user.

For instance, if a user presses a button in the View, a call is made to the Presenter from the View, which then processes the action. If necessary, it calls other methods in the Model in order to change the application state. When the Presenter has processed the action, it calls the View to present the result to the user.

11 Communication

As described in Chapter 6, BlueCAML consists of three packages: the mobile application, signaltransmitter, and the back-end. These packages needs to communicate with each other in order to update and maintain radio maps. This chapter describes how the communication is realised.

As described, following two events result in the need of communicating between the different parts of BlueCAML.

- The mobile application sends fingerprints, containing measurements from a number of signaltransmitters to the back-end through the smart signaltransmitters in order to update the radio map.
- The mobile application retrieves an updated radio map from the back-end by communicating through the smart signaltransmitters.

The following describes the message format used for the communication between the applications and uses model checking to verify a number of properties of the communication protocol.

11.1 The Communication Protocol

The communication protocol is as follows: the mobile application sends a request to the signaltransmitter, either requesting a radio map or sending corrections to it. Depending on the type of request, the signaltransmitter further communicates with the back-end, requesting the radio map or update it, respectively. In the case of the mobile application requesting the radio map, it is send back from the signaltransmitter. If the mobile application is correcting the radio map, it receives a response immediately. Afterwards, the signaltransmitter continues to send the corrections to the back-end.

When correcting the radio map, multiple calls need to be made between the signaltransmitter and the back-end. This is the case since RESTful interfaces specify that only one resource should be modified in a single request. Since the fingerprint and individual measurements,

comprising it, are different resources, these need to be added over multiple calls. Hence, to simulate that these comprise a single request, a transaction mechanism can be implemented. When the fingerprint is send, it sets the transaction to uncommitted. After all measurements have successfully been uploaded, the state of the transaction is set to committed, thereby signalling that the entire fingerprint is uploaded. Only fingerprints with committed state are afterwards considered.

11.2 Message Format

Communication between the signaltransmitters and the back-end is done using RESTful interfaces, meaning that signaltransmitters uses the HTTP protocol for communication.

In respect to the communication between the mobile application and the signaltransmitter, a customised message format is used. Figure 11.1 depicts this message format consisting of a 6 byte header and a variable length payload.

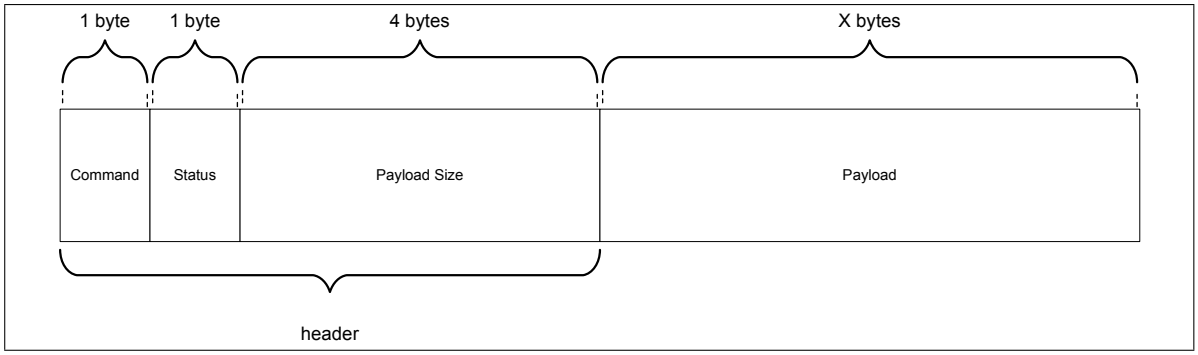


Figure 11.1: Header for the communication requests sent between the mobile application and the signaltransmitters.

As shown, the first byte indicates the specific command the mobile application requests handled. One byte is sufficient due to only two commands currently being supported in BlueCAML. When data is sent from the signaltransmitter to the mobile application, this field should contain the code for the requesting command even though it is not used by the mobile application.

The next byte indicates the status of the request. A status code of zero indicates success and all other status codes indicate a particular type of failure. The status is followed by a payload size field which is a four byte integer indicating the size of the following payload. The size is essential to know how much data should be received after the header.

11.3 Verification of Protocol

In Section 8.2 UPPAAL was introduced as a tool for statically verifying specifications or applications using model checking. In this section, it is described how the tool is used to verify different properties of the protocol.

The following describes the templates used in modelling the communication protocol. Notice that all channels used for synchronisation between the processes are made broadcast such that if a process is listening, it can synchronise, otherwise, the system continues.

The communication has been modelled using four templates: one for the mobile application, two for the signaltransmitter, and one for the back-end. The signaltransmitter is divided into two templates in order to model the communication between the mobile device and the back-end, respectively.

The time intervals have been determined experimentally by measuring the time it takes of simulating the operations. The time unit of the clock variables is in milliseconds.

11.3.1 Mobile Client

The `MobileClient` template, depicted in Figure 11.2, models the locations and edges of the `MobileClient`. It starts in the location marked *Idle* and from here, it can synchronise with the `StReceiver` by firing the only outgoing edge and signal *ConnectingSt*. When `StReceiver` has synchronised with `MobileClient` by signalling *ConnectedSt*, the `MobileClient` template can execute one of two different actions; it can either choose to send a fingerprint by signalling *SendFingerprintSt* or it can choose to get the radio map by signalling *GetRadioMapSt*. After conducting one of these two synchronisations, `MobileClient` enters respective wait locations in which it is able to wait for a total of 1000 and 5000 time units.

In case of getting the radio map, `MobileClient` times out if it has not within this time frame received either a time out signal from the back-end or a radio map.

In case of sending the fingerprint, `MobileClient` times out if it has not synchronised with the *SendFingerprintStatusSt* within the time frame. Finally, `MobileClient` enters the *Idle* location again and is ready to connect to `StReceiver`.

11.3.2 Signaltransmitter Receiver

The `StReceiver` template, depicted in Figure 11.3, models the process of the signaltransmitter handling the requests from the mobile client. As shown, it starts by connecting with a `MobileClient` process, and when connected, enters *ConnectedToMobile*. From there, it waits for the channel the `MobileClient` process synchronises with, depending on whether a radio map is requested or a fingerprint is sent. If synchronising with *GetRadioMapSt*, the process waits for the `Backend` process to be ready for requests. When it is, the request is passed on by signalling *GetRadioMapBackend*, and later when `Backend` signals on the *SendRadioMapSt* channel, synchronises with the `MobileClient` process. When waiting for `Backend`,

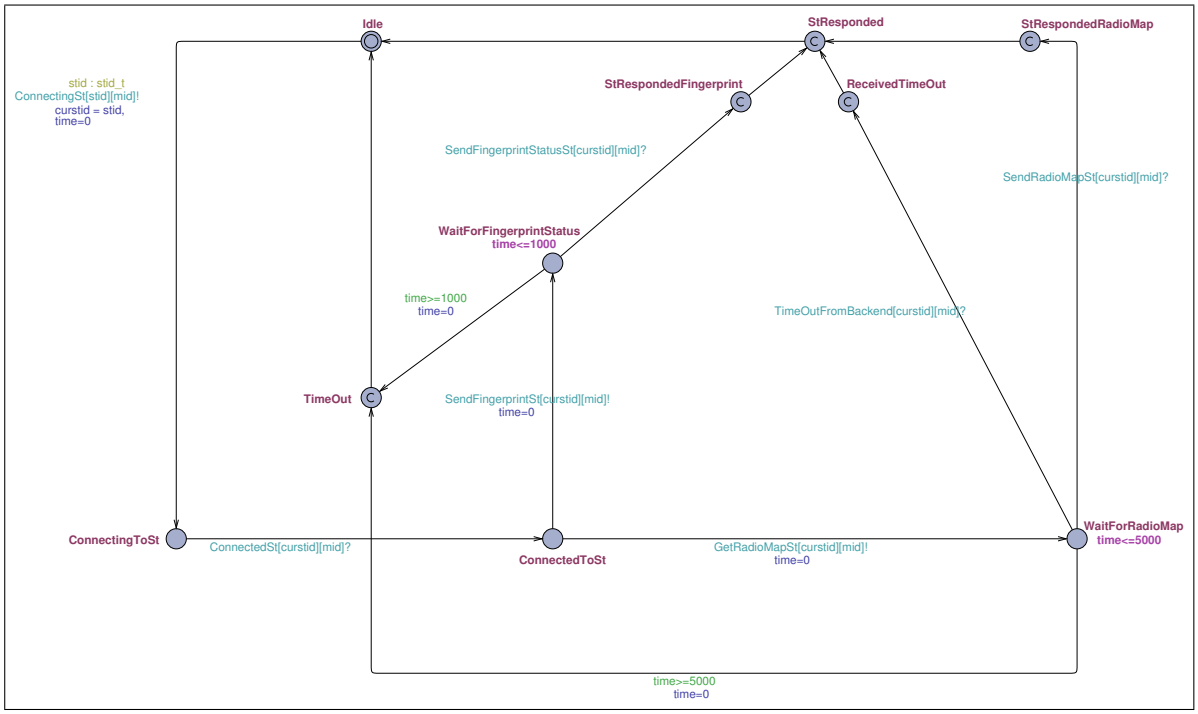


Figure 11.2: UPPAAL model for `MobileClient` and its communication with the `StReceiver` template.

the `StReceiver` process enters *TimeOutBackend* if it has been waiting for longer than 500 time units.

If synchronising with *SendFingerprintSt* from the *ConnectedToMobile* location, the model simulates that the received fingerprint either is in accordance with the expected format or not. In both cases, the *ReturnFingerprintStatusToMobile* location is reached. Only if the format is correct, a task is added to a queue used by `StSender`. From *ReturnFingerprintStatusToMobile*, the *SendFingerprintStatusSt* channel synchronises with the `MobileClient`.

11.3.3 Signaltransmitter Sender

Figure 11.4 depicts the `StSender` template. This models the process of continuously sending fingerprints from a queue to the back-end. This process is made in steps to simulate transactions in REST. Therefore, an uncommitted fingerprint is firstly send to the back-end, then the measurements are send, and finally the fingerprint is committed. If the fingerprint is not committed it will not be used in further processing by the back-end.

It starts in the location called *Idle* from which it, whenever the queue is non-empty, can start sending if the `Backend` process is ready for receiving. If the uncommitted fingerprint is send, the process enters the *WaitForBackendUncommittedFingerprint* location. Next, the *WaitForBackendUncommittedFingerprint* location is entered, and `StSender` waits for `Backend`

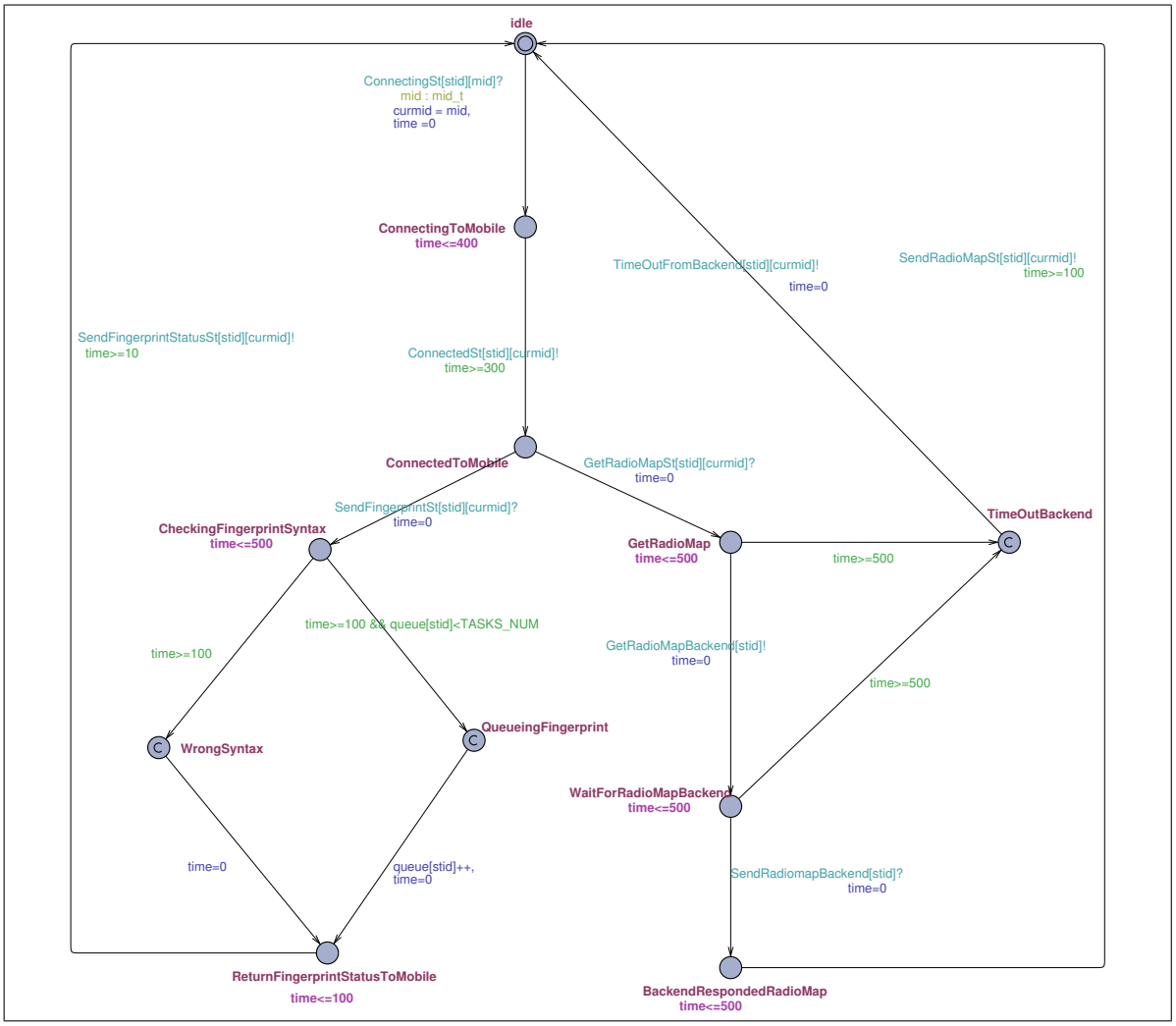


Figure 11.3: UPPAAL model for the `StReceiver` template which handles incoming requests from the `MobileClient` processes.

to respond. When the `Backend` responds, `StSender` enters the *ReadyToSendMeasurement* location. Here, sending the measurements is modelled and upon completion, the *WaitForBackendMeasurement* is entered until the `Backend` responds by synchronising with *MeasurementResponseBackend*. Finally, the fingerprint needs to be committed, modelled by synchronising with *SendFingerprintCommittedBackend* and entering the *WaitForBackendCommittedFingerprint* location waiting for response from `Backend`. Finally, the queue is decremented by one, indicating that the task is complete.

All the locations, depending on synchronisations with the `Backend` process, can time out. In the time out locations, the `StSender` process retries its task after backing-off for 2000 time units.

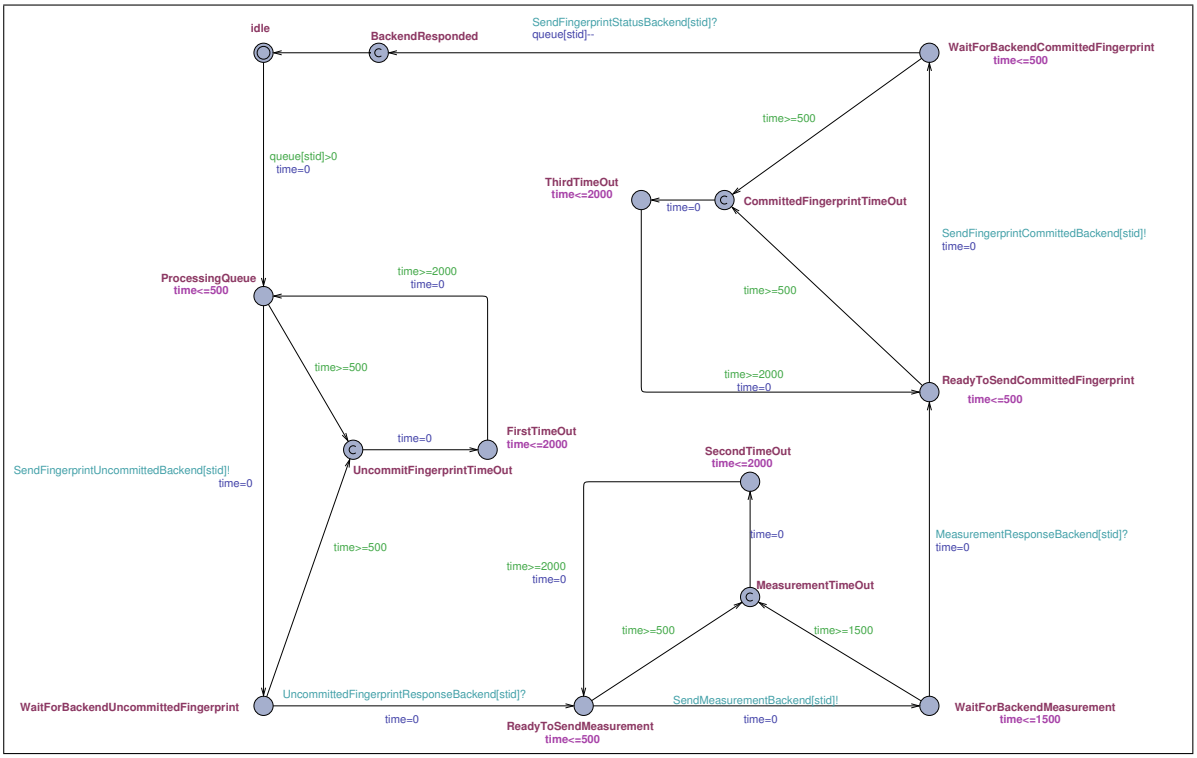


Figure 11.4: UPPAAL model for the `StSender` template which is responsible for sending requests from `StReceiver` to the `Backend`.

11.3.4 Back-end

The `Backend` template, depicted in Figure 11.5, models the back-end of the system. The template starts in the initial location marked *Idle*. From this location, it is able to synchronise with four different channels namely: *GetRadioMapBackend*, *SendFingerprintUncommittedBackend*, *SendMeasurementBackend*, and *SendFingerprintCommittedBackend*. All these channels represent the interactions available for the signaltransmitter. For each of the locations having an edge annotated with one of the previously described channels, an invariant is associated together with an outgoing edge on which a guard is placed. The invariant together with the guard represent how long the particular operation takes.

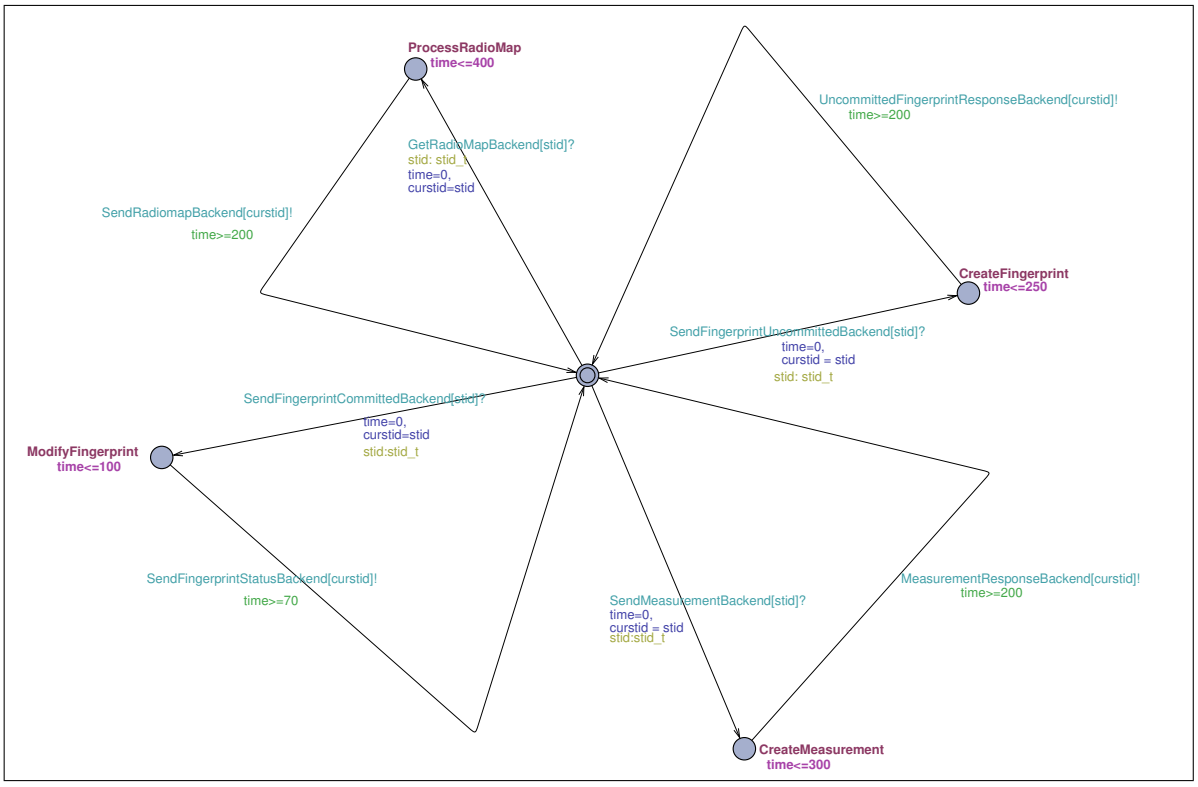


Figure 11.5: UPPAAL model of the `Backend` template communicating with `StSender`.

11.3.5 Verification

Using the described query language, certain properties of the model can be checked. Listing 11.1 shows an excerpt of the verified statements.

```
1  A[] !exists(i:mid_t)(MobileClient(i).TimeOut)
2  E<> exists(i:mid_t)(MobileClient(i).StRespondedFingerprint)
3  E<> exists(i:mid_t)(MobileClient(i).StRespondedRadioMap)
4  E<> StReceiver(0).ReturnFingerprintStatusToMobile
5  E<> StReceiver(0).BackendRespondedRadioMap
6  E<> StSender(0).BackendResponded
7  A[] StSender(0).UncommitFingerprintTimeOut imply StSender(0).time==500
8  A[] StSender(0).MeasurementTimeOut imply StSender(0).time==500 || StSender(0).time==1500
9  A[] StSender(0).CommittedFingerprintTimeOut imply StSender(0).time==500
10 A[] StReceiver(0).TimeOutBackend imply StReceiver(0).time==500
11 A[] !deadlock
```

Listing 11.1: Verified properties of the communication.

Generally, the model contains three `MobileClient` processes, two `StReceiver` processes, two `StSender` processes, and one `Backend` process. The scenario is chosen to simulate load on the signaltransmitters since there is one more mobile client than signaltransmitters, and likewise with the signaltransmitter and the back-end. The purpose is to verify whether the listed statements hold for this setup. The statements are described in the following.

- **Line 1** - Verifies that, when connected to the `StReceiver` process, the `MobileClient` process does not time out, meaning that the `StReceiver` always returns a response within the expected time.
- **Line 2** - Checks that at some point the `MobileClient` process gets a response when sending a fingerprint.
- **Line 3** - This verifies that, the `MobileClient` process at some point gets the requested radio map.
- **Line 4** - Checks that the `StReceiver` process eventually reaches the *ReturnFingerprintStatusToMobile* location indicating that the `StReceiver` responds to the `MobileClient` process. Similar statements are made for the other `StReceiver` processes.
- **Line 5** - Checks that the `StReceiver` class eventually reaches the *BackendRespondedRadioMap* location indicating that the `StReceiver` process responds to the `MobileClient` process. Similar statements are made for the other `StReceiver` processes.
- **Line 6** - Verifies that the `StSender` process eventually receives a response from the `Backend` process indicating that the fingerprint has been successfully stored. Similar statements are made for the other `StSender` processes.

- **Line 7** - This verifies that the clock variable equals 500 when the `StSender` process enters the *UncommitFingerprintTimeOut* location.
- **Line 8** - This verifies that the clock variable equals 500 or 1500 when the `StSender` process enters the *MeasurementTimeOut* location.
- **Line 9** - This verifies that the clock variable equals 500 when the `StSender` process enters the *CommitFingerprintTimeOut* location.
- **Line 10** - This verifies that the clock variable is equal to or above 500 time units when the `StReceiver` process receiver enters the *TimeOutBackend* location.
- **Line 11** - This verifies that the simulation does not reach a deadlock.

All the properties are concluded to be satisfied by UPPAAL. This means that if the code reflects the model, the properties also holds for the code.

Part III

Design

12 Back-end Application

The back-end application consists of a number of models used to store the radio map, and some RESTful interfaces used to access and maintain the radio map. In the following, the available web services, data design, and classes involved are described.

12.1 Web Services

The following web services are identified to fulfil the need of the signaltransmitters to get a radio map and send corrections to it. Their descriptions are part of this chapter since they pose an essential interface to the signaltransmitter that should be examined as clearly as possible to avoid that the signaltransmitter application, heavily relying on this interface, needs to be redesigned.

Get Radio Map This operation is supported through a single RESTful web service which allows the signaltransmitters to call it and get the radio map in XML format.

Send Corrections In order to support the signaltransmitter sending corrections to the fingerprint, a number of RESTful web services are made available. First, a fingerprint must be created, and a call must be made for each measurement which should be added to this fingerprint. Finally, a call is made to mark the fingerprint as committed.

All web service requests are handled by a controller, before using a view to render the response.

12.2 Data Design

In this section, the data design is described for the underlying models used in BlueCAML. Essentially, the data design conforms to the data structure of the Weighted Graphs algorithm and is responsible for storing relevant information that allows for constructing a radio map suitable for being stored on the mobile devices. The entity-relationship diagram modelling the database of the back-end, is shown in Figure 12.1. Using an entity-relationship for this

purpose is appropriate because the involved entities, attributes, and relationships can be directly mapped to a relational database which is used in the back-end.

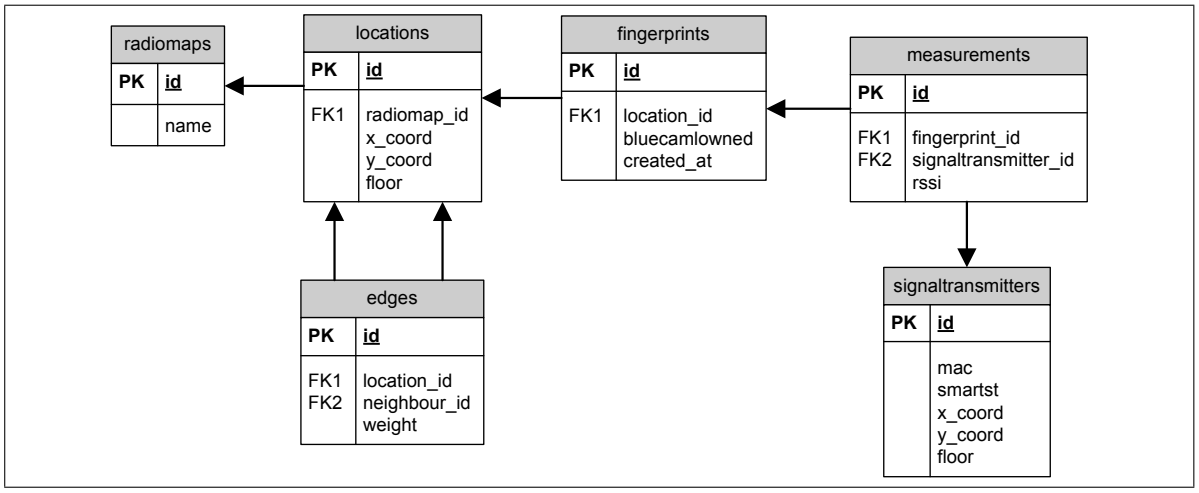


Figure 12.1: ER-diagram showing the relationships of the entities in the model of the back-end.

The *radiomaps* entity is responsible for storing the various radio maps BlueCAML is comprised of. This allows for having a centrally located back-end that governs multiple buildings. This entity is related to the *locations* entity with one-to-many cardinality describing all the locations the particular radio map is comprised of. In order to implement the Weighted Graph data structure, the *edges* entity relates a *location* with multiple other *locations* and in addition describes the weight of the edge, that is, the distance from two connected *locations*.

A single *location* can contain multiple *fingerprints* thereby allowing a history of *fingerprints* to be constructed. This is useful in regards to the maintainable part of BlueCAML where user-supplied fingerprints need to correct the currently stored fingerprints. The *bluecamlowned* attribute, indicates whether or not the particular fingerprint has been conducted as part of the official measurements.

As previously described, a fingerprint is comprised of multiple RSSI measurements from a number of signaltransmitters. To make this relationship in the model, the *measurements* and *signaltransmitters* entities are used. The *measurements* entity contains an average RSSI measurement from the particular signaltransmitter to which it is related in the model. Finally, the *signaltransmitters* entity contains information about the signaltransmitters in the radio map which includes the Bluetooth MAC-address, a boolean value describing whether or not the particular signaltransmitter is smart, and finally the coordinates of its location. The coordinates are not currently being used, but they have been included due to the premise that they might be convenient in a deployment situation where maintainers need to determine where specific signaltransmitters are located in the environment.

12.3 Classes

The purpose of this section is to describe the classes involved in the back-end. It has been assessed that including this design document is convenient, because it provides a good overview of the entire application. Hence, this design document is included for all constituents of BlueCAML due to its importance.

The documentation of the design of the back-end application is not based on a UML class diagram due to the classes not being associated with each other directly. Only the models of the MVC architectural design pattern are interrelated, but this relationship can be directly transposed to the previously described data design. Due to this, the classes are simply represented in their respective component of MVC which can be seen in Figure 12.2.

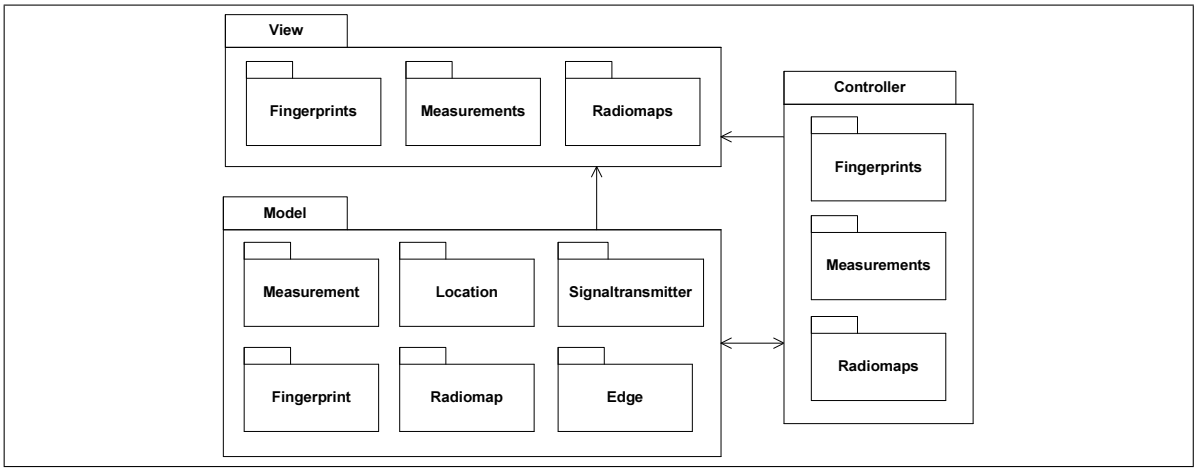


Figure 12.2: Classes involved in the back-end. The figure shows to which component they belong in the MVC architectural design pattern. The arrows indicate how information is passed among the constituents of MVC.

In the following, an accompanying description of each of the classes in the view and the controller is given.

View

Fingerprints This class is responsible for rendering the view of the *Fingerprint* entity of the model. With regards to the signaltransmitter, this involves representing the attributes of *Fingerprints* in XML.

Measurements Similar to the previous class, this class renders the *Measurements* entities in XML.

Radiomaps This view is responsible for rendering and entire radio map as XML. This is necessary when the mobile devices request radio maps.

Controller

Fingerprints This controller accepts incoming HTTP request with the HTTP methods POST and PUT. The POST request, is with respect to the principles of RESTful interfaces, used for creating a new resource as described in Section 9. This means that when a mobile device uploads a fingerprint, a POST request will be issued. When the previously mentioned POST has been issued, the fingerprint is considered uncommitted. When *Measurements* have been uploaded as well and connected to the uncommitted fingerprint, the PUT request is performed, thereby, in terms of RESTful interfaces, signalling that the resource should be updated to be committed.

Measurements The *Measurements* controller accepts incoming POST HTTP requests which are used to create new RSSI measurements when a mobile device uploads a fingerprint.

Radiomaps This controller implements functionality when a HTTP GET request is issued to the web server. This indicates that a resource is requested for retrieval in terms of RESTful interfaces. In this case the entire radio map is retrieved.

13 Signaltransmitter

The following chapter documents the design of the signaltransmitter application. This is done by describing the classes of it and how they interrelate. Furthermore, the activity of uploading a fingerprint to the back-end is described in detail.

13.1 Classes

This section serves to describe the design of the signaltransmitter application in terms of its classes. For this purpose, a UML class diagram is used and the particular class diagram of the signaltransmitter application is depicted in Figure 13.1.

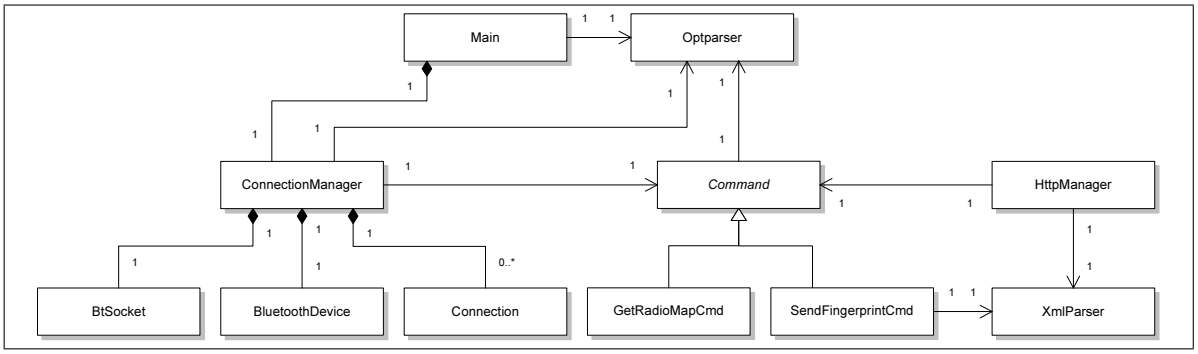


Figure 13.1: Class diagram depicting the structure of the signaltransmitter.

In the following, a description of each of the classes is provided.

Main The `Main` class is responsible for initialising the application. This comprises setting up the environment with the `OptParser` class and subsequently initialising `ConnectionManager` to accept and process incoming Bluetooth requests.

Optparser This class is responsible for setting up the environment and acts as a helper class for `Command` and `ConnectionManager`. Doing this, comprise parsing the command line arguments provided to the application. The `Optparser` class is based on the Singleton

design pattern to restrict that only one instantiation of the class to be possible. Whenever an object is instantiated, a reference to the same object will be retrieved containing the environment just described.

ConnectionManager The `ConnectionManager's` area of responsibility comprises setting up a Bluetooth RFCOMM socket on the local Bluetooth device listening for incoming requests from mobile devices. A request is processed by means of using the corresponding derived class of the `Command` class.

BtSocket This class is responsible for creating an RFCOMM socket listening for incoming requests on the local Bluetooth device.

BluetoothDevice The `BluetoothDevice` class represents the local Bluetooth device in the signaltransmitter. Hence, it contains the MAC address of the device and provides access to it.

Connection This class is used for representing the connection between the signaltransmitter and the mobile devices. Besides this representation, it offers the possiblity of writing and reading to and from the client socket. Hence, it is capable of reading and writing header information dictated by the protocol and for processing the payload accordingly.

Command This abstract class is used for providing a means for the `ConnectionManager` class to use the supported commands through polymorphism. The subclasses of the `Command` class act as a proxy between the mobile devices and the back-end when, for instance, the mobile devices requests the radio map. There are two subclasses of this class which are described below.

GetRadioMapCmd This concrete class of the `Command` class implements the functionality necessary for retrieving the radio map from the back-end and subsequently send it to the requesting mobile device.

SendFingerPrintCmd This concrete class of the `Command` class, implements functionality concerning sending a fingerprint to the back-end. For doing this, it first processes the incoming XML from the client and extracts the location of which the particular fingerprint was conducted and additionally extracts RSSI measurements with corresponding signaltransmitter. Afterwards, the `SendFingerPrintCmd` class passes this information to the `HttpManager` which is responsible for queuing the REST HTTP request thereby caching it and processing it when available.

HttpManager The `HttpManager` class contains functionality that allows for interacting with a web server with RESTful interfaces. In the case of sending a fingerprint to the web server, the `HttpManager` contains some dedicated logic that allows for queuing HTTP requests. In this relation, the `HttpManager` places the request in a queue and a dedicated thread is responsible for continuously processing this. Finally, the `HttpManager` is based on the Singleton pattern to allow only a single instance of the class to be created.

XmlParser For subclasses of the `Command` class, the response from the web server and requests from the mobile devices are represented as XML which need to be parsed. The `XmlParser` class is responsible for conducting in-memory processing of the XML through the evaluation of XPath expressions.

13.2 Upload Fingerprint Activity

The purpose of this section is to document the activity of uploading a fingerprint from the signaltransmitter to the back-end utilising its RESTful web services. Hence in the following, it will be assumed that the mobile device, prior to this activity, has successfully uploaded a valid fingerprint represented as XML, parsed it, and, finally, the RSSI measurements have been extracted. The final step is to construct an element that represents the particular HTTP request for the fingerprint and enqueue it into a shared queue data structure. In Figure 13.2, the UML activity diagram of the queue processor is depicted.

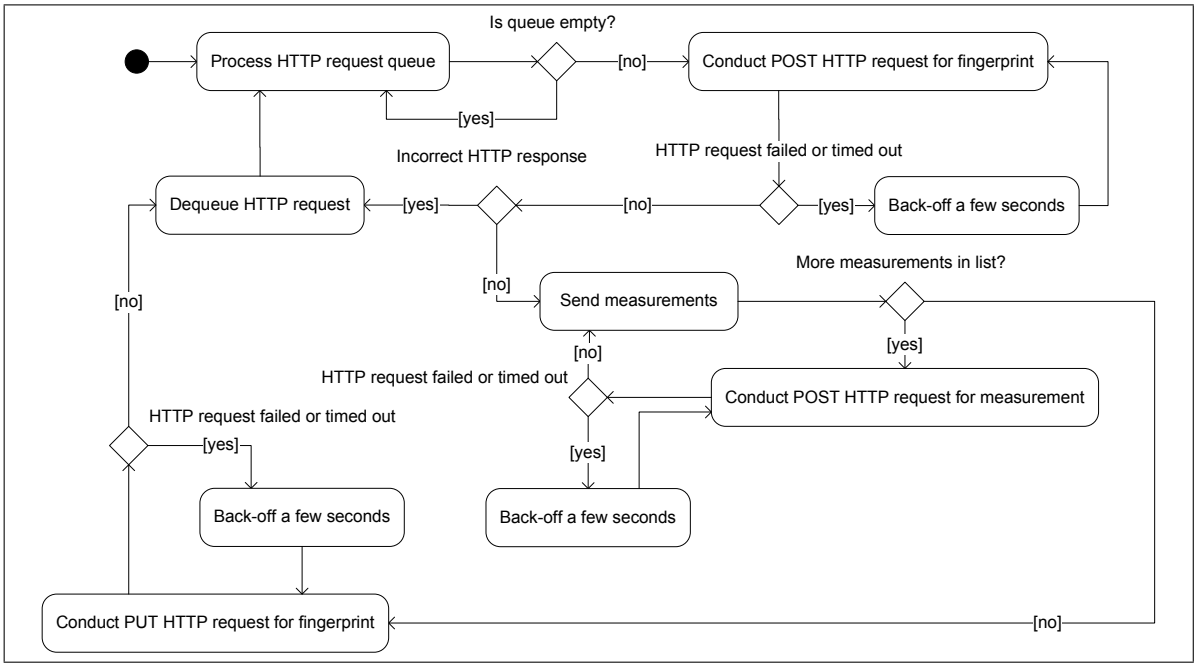


Figure 13.2: UML activity diagram showing how the signaltransmitter interacts with the back-end to send a fingerprint.

The activity takes starting point in investigating whether or not the queue contains HTTP request elements that need to be processed. If there are any, the next step is to conduct a POST HTTP request to the back-end to the fingerprint resource. According to the model outlined in Chapter 11, the signaltransmitter should keep conducting the HTTP request until it is successfully received at the back-end. This means that in case it fails, it will back-off a few seconds and thereafter try again. In the subsequent HTTP request, the same procedure

will be assumed. The corresponding HTTP response from the HTTP POST contains the newly created fingerprint instance comprising all its attributes including the id represented in XML.

The id is necessary for constructing a relationship between the fingerprint and the measurements entities. Subsequently, a loop is performed that conducts HTTP POST requests to the back-end for each RSSI measurement. Whenever a measurement is posted, the fingerprint id to which it belongs is additionally provided.

The final step is to conduct a HTTP PUT request to the newly created fingerprint. This is used for setting the fingerprint into the committed state. If this request is not successfully conducted, the uploaded fingerprint will not be considered complete and will be omitted in later processing by the back-end. Given that the HTTP PUT request was conducted successfully, the HTTP request element will be dequeued from the queue and the loop starts over again by processing the next request.

14 Mobile Application

This chapter documents the packages which the mobile application consists of, and how they interact with each other. Furthermore, the activities of the location engine and the maintenance engine are described.

14.1 Classes

Figure 14.1 depicts the class diagram for the mobile application. As shown, the mobile application consists of five packages which consist of multiple classes. The packages and classes are described in the following.

As previously described the mobile application is based on the MVP design pattern.

Presenter

This package consists of a single class responsible for handling all user input.

Presenter The `Presenter` class links the view with the model by handling all user interaction. Also, it interacts with the model to change the application state and interacts with the View Layer to show changes in the application state to the user.

View Layer

The classes in this package are responsible for all user interface related operations.

View The `View` class creates a Windows Form which is configured with all user interface related elements. Therefore, the class is responsible for representing information to the user.

PIFC Renderer The `PIFC Renderer` is responsible for rendering the map on the GUI. It is a component which contains a number of classes in order to handle both the PIFC file format and rendering of a map. The `View` class is responsible for creating and controlling this component.

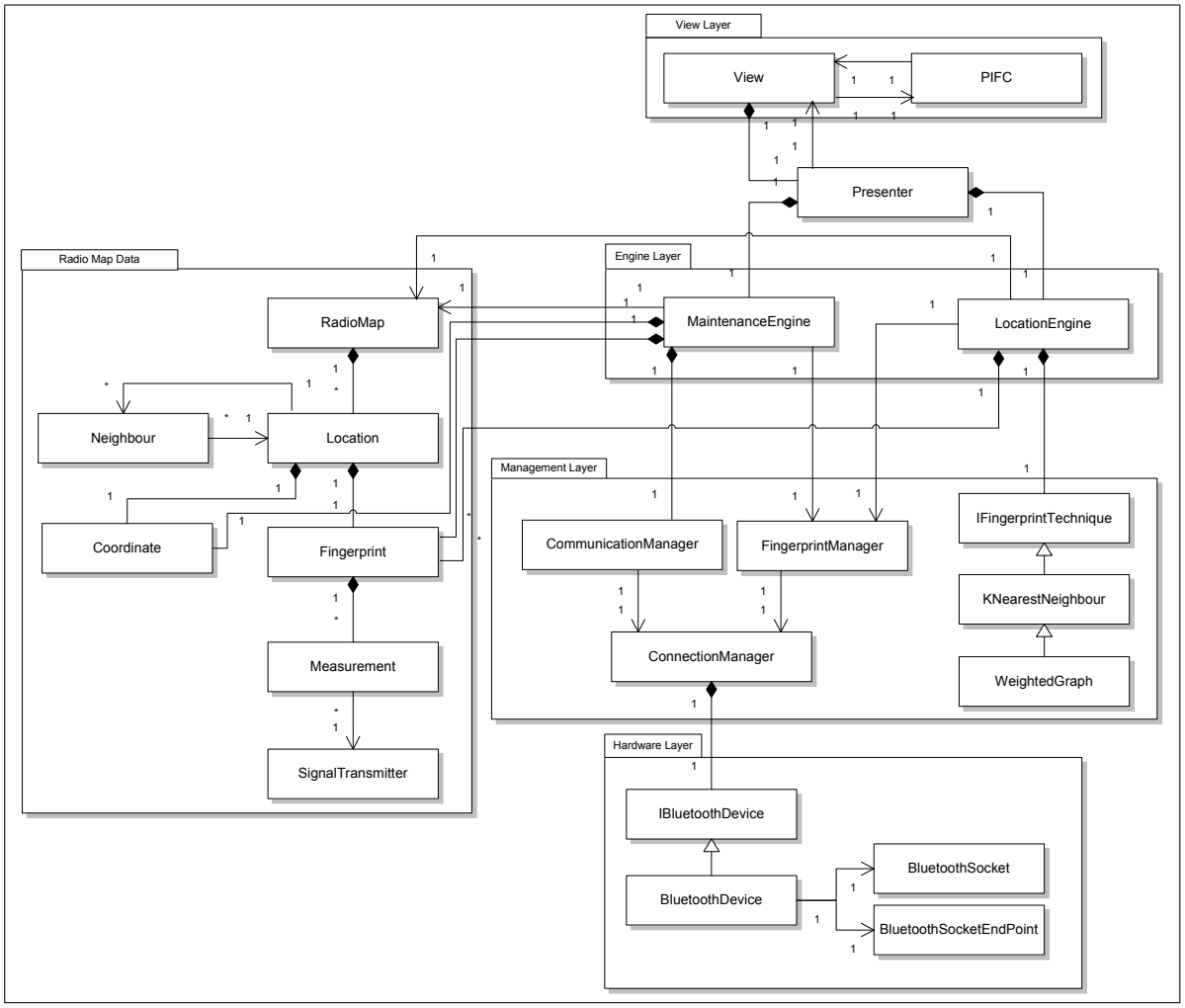


Figure 14.1: Class diagram depicting the structure of the mobile application.

Engine Layer

The classes contained in this package are responsible for estimating the location of the user and for providing the functionality in relation to maintaining the radio map.

LocationEngine The area of responsibility of this class comprises conducting a fingerprint by using the `FingerprintManager` and then, by utilising a specific strategy, estimate the position of the user by comparing the conducted fingerprint with the fingerprints contained in the radio map.

MaintenanceEngine The purpose of this class is to provide the user with the functionality of maintaining the radio maps. Thus, it must conduct fingerprints by using the `FingerprintManager` in four directions at a given location and send the final fingerprint to a smart signaltransmitter.

Management Layer

The purpose of this package is to provide business logic to the two engines in the Engine Layer.

CommunicationManager This class is responsible for all data transmission between the mobile application and the back-end through the signaltransmitters. It uses the `ConnectionManager` to find an appropriate signaltransmitter it can communicate with.

ConnectionManager This class is responsible for maintaining a list of connected signaltransmitters. This is used by the `FingerprintManager` when measuring RSSI values and by the `CommunicationManager` when selecting a signaltransmitter for data transmission.

FingerprintManager The `FingerprintManager` is capable of measuring RSSI values from multiple signaltransmitters and construct fingerprints which is used in position estimation and maintenance.

IFingerprintTechnique Since research continuously is made in fingerprint techniques, it is expected that the used technique might need to be changed over time. Therefore the current technique is implemented using this interface making the process of replacing it relatively easy. The classes implementing this interface are described below.

- KNearestNeighbour In Chapter 5, it was determined that the KNN fingerprint technique is used. This class is the implementation of that technique. The class inherits from the `IFingerprintTechnique` class such that it can be used as a technique in the `FingerprintManager` by using polymorphism.
- WeightedGraph As previously described in Chapter 5, the Weighted Graphs fingerprinting technique is used in the mobile application. This class is the implementation of that technique. Notice that it inherits from the `KNearestNeighbour` class. This is done since it is a set of extra ways to optimise the KNN position estimation technique.

Radio Map Data

The classes in this package represent the data related to the radio map which must be handled in the mobile application. This includes locations, signaltransmitters, and measurements. These classes are only used to represent the data, and all data manipulation is done by the Management Layer.

RadioMap This class reflects the data used to represent a radio map, and references the locations and signaltransmitters in it. The class is based on a Singleton design pattern which ensures that only one instance of a radio map exists that can be shared among the different classes.

Location A `Location` is a combination of a location in the building, represented with a `Coordinate`, and an associated `Fingerprint`. Furthermore, the location refers to other locations through the `Neighbour` class in order to specify its neighbouring locations.

Neighbour This class references `Location` and the distance to it from the `Location` which owns the `Neighbour`. The `Location` class could contain this information, but for code readability, it has been split into its own class.

Coordinate This represents a specific point in the building in x, y, and floor coordinates. This class is used throughout the application when representing a physical location in the building.

Fingerprint This represents a fingerprint which contains a list of `Measurements` associated with a specific `Signaltransmitter`.

Measurement The `Measurement` class represents an actual measurement to a signaltransmitter, containing an averaged RSSI value.

SignalTransmitter The signaltransmitters are represented using this class, containing their MAC-address, id-number, and an indication of whether they are smart or dumb.

Hardware Layer

The purpose of this package is to provide an abstraction layer above the hardware implementation of the Bluetooth device. This way, the other classes do not need to be aware of the specific hardware implementation.

IBluetoothDevice The purpose of this interface is to increase testability since a dummy Bluetooth device can be used as long as it fulfils the interface. The classes implementing this interface are described below.

BluetoothDeviceWM6 This class handles communication with the Bluetooth device.

BluetoothSocket This class is responsible for establishing an RFCOMM socket on the local Bluetooth adapter of the mobile device.

BluetoothSocketEndPoint The class is used in regard to establishing an RFCOMM socket connection to a remote Bluetooth device. The specific code in this class is inspired by Microsoft code samples (Microsoft, 2010).

14.2 Activities

In the following, two different activities in the mobile application are described in further detail. These are the main operations of the location engine and the maintenance engine. They are shown since, they represent central parts of the mobile application.

For the purpose of visualising the activities, UML activity diagrams have been used. The convenience of these is evident in that they provide a good overview of relatively complex or essential procedures that need more emphasis to avoid subsequent errors. In this regard, an improved understanding is obtained or so far undiscovered functionality identified.

14.2.1 Location Engine

The main process of the location engine can be divided into eight steps. Of these, two are related to the `ConnectionManager` and three are related to position estimation. Figure 14.2 shows an activity diagram of the location engine when conducting position estimates.

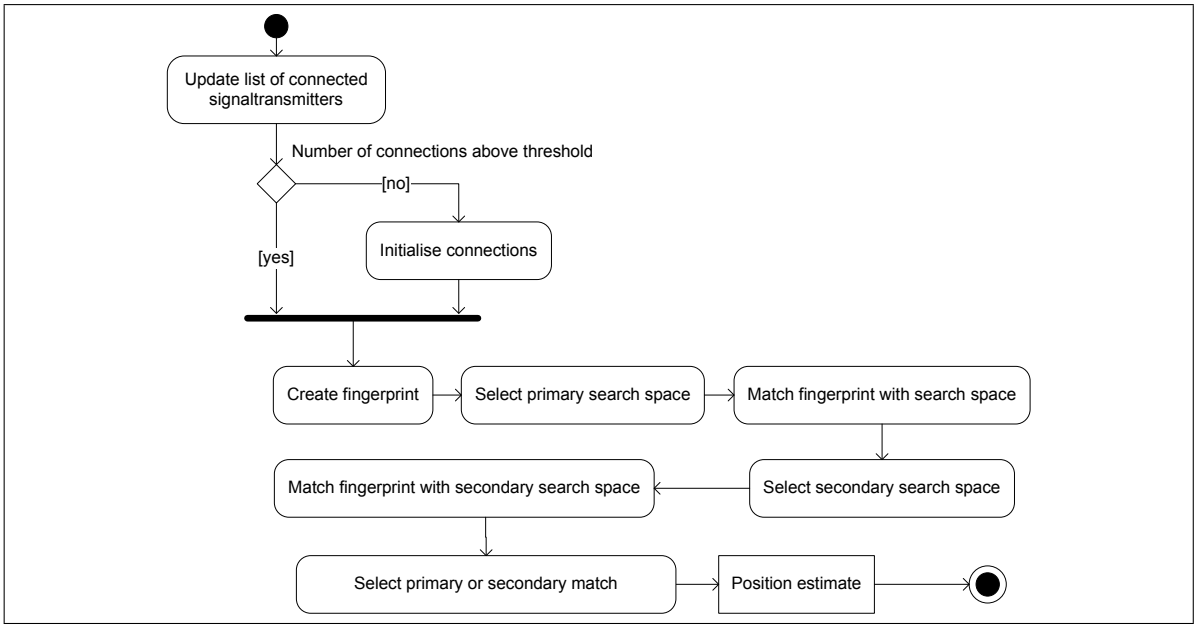


Figure 14.2: A UML activity diagram showing the main process of the location engine concerned with creating position estimates and invoking the `ConnectionManager` when appropriate.

The location engine's first task is to invoke the `ConnectionManager` in order to update the list of currently connected signaltransmitters. The `ConnectionManager` then removes or adds new connections as appropriate within certain time-limits. If the number of connected signaltransmitters is below a certain threshold, it is estimated, that the position estimate is too inaccurate and the location engine invokes the `ConnectionManager` again to search for all visible signaltransmitters. For instance, if only one signaltransmitter is detected, it is difficult to make a match in the radio map.

Since it is desirable to reduce the number of times the initialisation operation is conducted, because it can be time consuming, the threshold is set to two based on experiments. Also, an upper bound of concurrently connected signaltransmitters is given. This is based on the fact that in the offline stage, no more than four signaltransmitters could be detected simultaneously at a location, meaning that trying to connect to a fifth, simply would be waste of time.

When the current connections have been maintained, six actions are conducted to get a position estimate. First, a fingerprint is created by invoking the fingerprint manager which in turn measures the RSSI values of connected signaltransmitters. Then, the primary search space is selected and the fingerprint is matched with this. Afterwards, the secondary search space is selected and the fingerprint is likewise matched with this. Before estimating the position, either the result of the primary of secondary search space is chosen. The estimated position is then based on this match.

14.2.2 Maintenance Engine

The purpose of the maintenance engine is to create fingerprints for specific locations and upload them to the back-end in order to maintain the radio map. This process involves a number of steps which continuously prompts for confirmation from the user. The actions required for carrying out the operation of successfully collecting a fingerprint are shown in Figure 14.3.

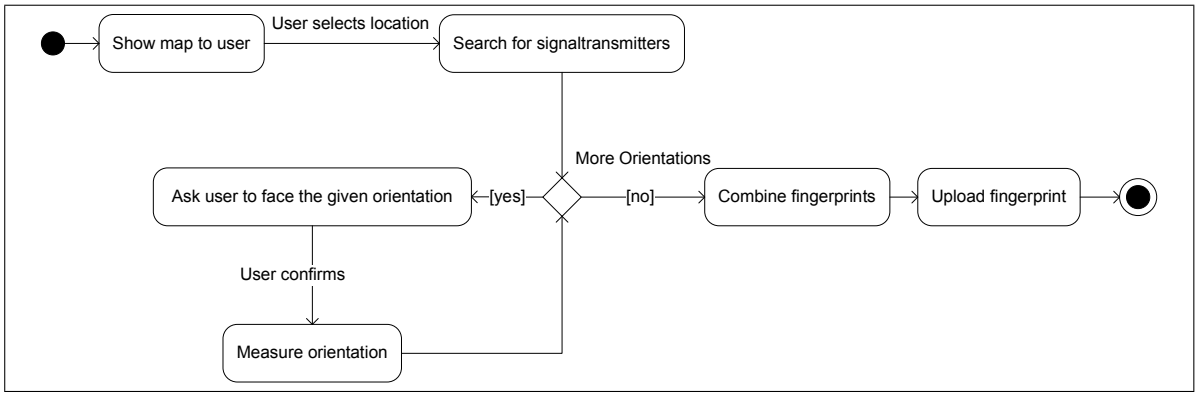


Figure 14.3: A UML activity diagram showing the actions involved in maintaining a single location.

Initially, a map is shown to the user on which locations having associated fingerprints are shown. The user selects a location and is instructed to wait while a discovery search is used to determine all visible signaltransmitters at the current location. The user is then asked to face north, east, south, and west and constructs fingerprint. These four fingerprints are then merged, calculating the average of the measured RSSI values, and uploaded to the back-end which is then responsible for further processing.

Throughout this process, the user is allowed to cancel the operation or the maintenance engine cancels the process itself if it determines that it is unable to find any signaltransmitters. This has not been shown in Figure 14.3 for verbosity reasons.

Part IV

Implementation

15 Back-end Application

Most of the models, views and controllers in the back-end application have been generated by the Ruby on Rails framework using its scaffolding system. One of the interesting parts of the back-end applications is the construction of a radio map based on the data in the database. Specifically, how corrections to the radio map are taken into account will be focused on during this chapter. A detailed explanation of this implementation is given in the following.

15.1 Construction of Radio Map

The radio map is constructed using the stored locations, fingerprints, and signaltransmitters which are combined into an XML document. When the users maintain the radio map, they associate new fingerprints with the existing locations such that multiple fingerprints are associated with them. However, for simplicity, only one fingerprint is stored for each location when transferred to the mobile device. The advantage of this approach is that if the strategy of combining the correction fingerprints is changed, then it can be done server side and propagated to the users when they update their radio map. Otherwise, it would require the users to update their mobile application.

As described in Chapter 7, there exist two types of fingerprints in the system: official fingerprints and user corrections. The procedure is to either return the newest official fingerprint or the average of two consecutive approximately equal corrections, made after the last official fingerprint.

Listing 15.1 shows how the *choose_fingerprint* method is defined, which is responsible for selecting the actual fingerprint used for a given location in a radio map. The method is placed in the `Location` model.

```
1  def choose_fingerprint
2      official_fingerprint = fingerprints.find(:last,
3        :conditions => {:bluecamlowned => true, :committed => true},
4        :order=>'created_at ASC')
5
6      if official_fingerprint == nil
7          return nil
```

```
8     end
9
10    corrections = fingerprints.find(:all,
11      :conditions => ["created_at>? AND committed=?",
12                      official_fingerprint.created_at, true],
13      :order=>'created_at DESC')
14
15    corrections.each_with_index do |correction, index|
16        if index+1 < corrections.length
17            if has_same_signaltransmitters(correction, corrections[index+1]) and
18              correction.distance_to_fp(corrections[index+1]) < 9
19                return get_average_fingerprint(correction, corrections[index+1])
20            end
21        end
22    end
23
24    return official_fingerprint
25 end
```

Listing 15.1: The method *choose_fingerprint* is responsible for selecting the best fingerprint for a given location.

First, in line 2-4 the last committed official fingerprint for the given location is found. The *fingerprints* belong to the current location and contains fingerprints associated with it.

In line 6, a check is made, which ensures that at least one official fingerprint exists, otherwise the method returns nil. This assumes that not all locations have an associated official fingerprint.

On line 10, a list of corrections is retrieved by selecting all committed fingerprints created after the last official fingerprint. These are placed in a list with the newest first.

In line 15, the list is then iterated, first checking if there exists one more correction in the list, shown in line 16. If it does, the current and consecutive corrections are compared in line 19 and 20. If they contain the same signaltransmitters and have a distance below 9 then the average of the two are returned. The distance threshold of 9 is derived from the assessment that the RSSI values for four signaltransmitters at maximum must deviate with three RSSI values, giving a Manhattan distance of 9. In an actual deployment, this number should be adjusted based on experiments.

16 Signaltransmitter

The signaltransmitter application comprises several essential components for enabling the proxy-like behaviour between the mobile device and the back-end. It contains two threads of execution: One which is responsible for accepting and processing incoming requests on the RFCOMM socket and one which is responsible for continuously processing the HTTP request queue. Multi-threading is based on an implementation of the POSIX thread (pthread) library.

When the first thread accepts an incoming connection from a mobile device, it starts by reading the number of bytes equivalent to the header of the communication message format. It starts by extracting the command requested by the mobile device which can be either the retrieval of the radio map or that a fingerprint is requested for upload. In addition, it extracts the payload size from the header information and reads the additional bytes corresponding to this number from the RFCOMM socket and buffers it for further processing.

The following describes the implementation of the activity of uploading a fingerprint to the back-end. This activity is chosen since it represents one of the interesting parts of the signaltransmitter.

16.1 Upload Fingerprint

When a request is made from the mobile device to the signaltransmitter, it performs a lookup in a hash table which hashes a command to the corresponding `Command` subclass. Using polymorphism, the process function on this is then called. In case of retrieving a radio map, the signaltransmitter performs a HTTP GET request on the radio map resource of the back-end and writes the HTTP response back to the client. In case of the desire to upload a fingerprint, the signaltransmitter parses the payload as an XML document, by using *libxml2*, and extracts the relevant information such as RSSI measurements and location from which the fingerprint was conducted through the utilisation of XPath evaluations. The extracted data is placed in a suitable element representing the desired HTTP request to the back-end, and requests the `HttpManager` to conduct the HTTP request. The `HttpManager` signals that it has successfully received the request which indicates that the signaltransmitter can respond back to the mobile device with status field of the header set to zero.

The `HttpManager` abstracts over a queue in which the aforementioned HTTP requests are placed. The second thread of execution processes this queue by means of the *http_req_processor* function. In the following the implementation details of this function will be given.

16.1.1 Processing HTTP requests

In Listing 16.1, an excerpt of the *http_req_processor* is given.

```
1  void *Httpmanager::http_req_processor(void *t_arg) {
2
3      ...
4
5      for (;;) {
6          pthread_mutex_lock(&process_http_req_mutex);
7
8          if(empty_queue())
9              pthread_cond_wait(&process_http_req_cond, &process_http_req_mutex);
10
11         Http_post_fp cur_http_post = get_front_queue();
12
13         meas_list = cur_http_post.get_meas_lst();
14         fp_res_url = cur_http_post.get_fp_res_url();
15         loc_id_str = cur_http_post.get_fp_loc_id();
16
17         post_field_str = construct_fp_post_fields(loc_id_str.c_str());
18         do {
19             memset(post_resp_buf, 0, http_post_buf_size);
20             http_ret_code = post(fp_res_url.c_str(), post_field_str.c_str(), &
                   post_resp_buf);
21             if(http_ret_code == Httpmanager::FAIL || http_ret_code == Httpmanager::
                   TIMEOUT)
22                 sleep(http_err_sleep_time);
23         } while(http_ret_code != Httpmanager::SUCCESS);
24
25         ...
26 }
```

Listing 16.1: The *http_req_processor* is responsible for processing the FIFO queue of HTTP requests.

As shown, the *http_req_processor* runs in an endless loop. In this, it initially blocks the calling thread, by calling the *pthread_cond_wait* routine. It will perform this call when the queue is empty. *http_req_processor* will be blocked until the *process_http_post_cond* condition variable is signalled which is done whenever a HTTP request element is placed in the queue as a consequence of a mobile device wanting to upload a fingerprint. The reason for using a condition variable is that it provides a means for threads to synchronise based upon the actual value of data. Alternatively, the thread executing the *http_req_processor* function

could continuously poll to check whether it has become non-empty. Polling is generally resource consuming since the thread would always be busy conducting this check. This is especially critical in terms of the signaltransmitter which contains limited resources that should be preserved whenever possible as this might possibly negatively influence the thread responsible for reading from the RFCOMM socket.

When the condition variable has been signalled, the next step in the function is to retrieve the front element of the HTTP request queue, line 11. From this element, the linked list of RSSI measurements is extracted together with the resource URL of the fingerprint and the location id from which the fingerprint was conducted. Given the location id, the POST fields of the subsequent POST of the fingerprint are constructed. This comprises setting the location id belonging to the fingerprint, setting that the fingerprint is currently uncommitted as part of the transaction, and finally set that the uploaded fingerprint is from a user in the post fields.

The last step of the excerpt is entering a do-while loop in which the *http_req_processor* continuously tries to conduct a POST request based on the URL transfer library *libcURL*. The POST request is performed to the URL constructed from a base URL and the fingerprint resource part. When the `HttpManager` class was instantiated, the base URL was provided and the fingerprint resource part is simply concatenated to this. When the response code from the HTTP request is *SUCCESS*, the loop breaks and the *http_req_processor* continues in a similar manner with conducting HTTP POST request for each of the RSSI measurements in the *meas_list* linked list and finally commit the transaction with a HTTP PUT request to the newly created fingerprint.

17 Mobile Application

In order to keep the user interface responsive, the application is based on multiple threads of execution. One thread is always used for the user interface, denoted the UI thread. This thread is used for rendering the user interface and reacting to user interaction. The `Presenter` class ensures that all actions using the model and potentially could make the user interface unresponsive are executed in a worker thread. Thus, when the `View` calls the `Presenter`, it creates a worker thread which further processes the request, and invokes back into the UI thread in order to update the View Layer.

In the following, the implementation details are described for the two central parts of the application. These are how the `LocationEngine` estimates a position and how the `MaintenanceEngine` is used to maintain radio map information for a specific location. Furthermore, notes are provided on changes made to the reused PIFC Renderer.

17.1 Position Estimation

The UML sequence diagram has been adopted because it provides a good overview of concrete functionality which is to be implemented. In addition, the usage of sequence diagrams has been assessed appropriate because they provide a description of a procedure on an abstraction level that can relatively easy be converted into the actual implementation.

The main method of the `LocationEngine` is the *`UpdateLocation`* method which is responsible for position estimation. Figure 17.1 depicts a UML sequence diagram of calling *`UpdateLocation`*.

As shown, the `LocationEngine` starts by calling *`UpdateConnectedSignalTransmitters`* on the `BluetoothConnectionManager` with the fingerprint from the last measurement. Using this information, the `BluetoothConnectionManager` maintains a list of connected signaltransmitters by establishing a connection to those that should be detectable in the given area. The connections are made using the *`Connect`* method of the `BluetoothDeviceWM6` class. Also, a time limit is taken into account such that only a relatively small number of connections are tried to be established. The purpose of doing this is to keep the update interval as low as possible by continuously connecting to new signaltransmitters.

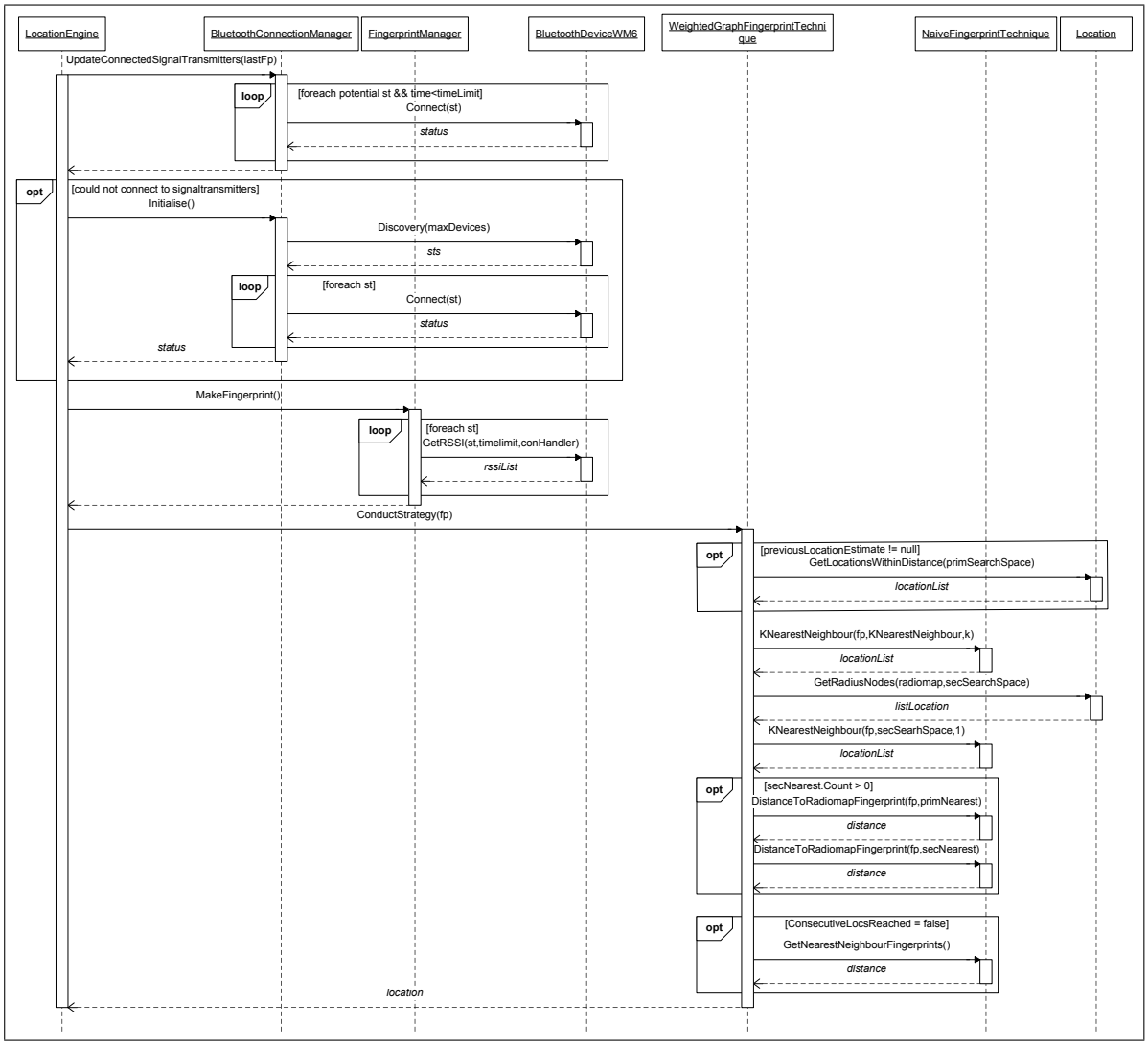


Figure 17.1: UML sequence diagram depicting the classes and method invocations involved in the *UpdateLocation* method.

If the *UpdateConnectedSignalTransmitters* is not able to maintain a list of connected signaltransmitters of a predefined size, an initialising phase is started by calling *Initialise* on the `BluetoothConnectionManager`. In this phase, the `BluetoothDeviceWM6` class first makes a discovery scan and afterwards establishing connections to the discovered signaltransmitters are tried.

Having a set of connected signaltransmitters, the `FingerprintManager` is used to measure RSSI values from each of these by using the *GetRSSI* method from the `BluetoothDeviceWM6` class. This fingerprint is then used in the final step where the *ConductStrategy* method of the `WeightedGraphTechnique` class is used to estimate the position of the user.

Figure 17.2 shows a screenshot of the mobile application when presenting a position estimate to the user.

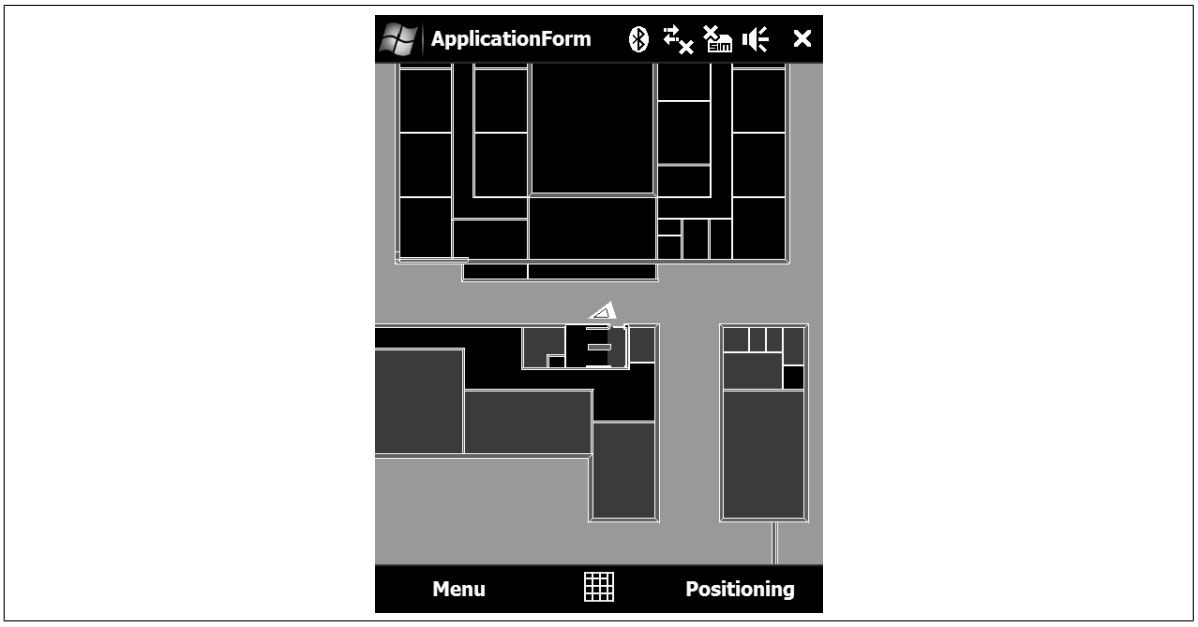


Figure 17.2: Screenshot of the mobile application when a position estimate is presented to the user. The white triangle indicates the currently estimated position.

In the following, the *ConductStrategy* is described since it comprises the most interesting functionality of the `LocationEngine`.

17.1.1 ConductStrategy

The *ConductStrategy* method of the `WeightedGraphFingerprintTechnique` class performs the Weighted Graphs fingerprint technique given the fingerprint collected during the online stage and the radio map. In Listing 17.1, the implementation of this method is shown.

```
1  public override void ConductStrategy(Fingerprint fingerprint)
2  {
3      if (this._bestMatchingLocation == null)
4          _primSearchSpace = this._radioMap.Locations;
5      else
6          _primSearchSpace = this._bestMatchingLocation.GetLocationsWithinDistance(this.
               _primSearchSpaceDist);
7
8      List<Location> primaryNearestLocations = KNearestNeighbour(fingerprint,
           _primSearchSpace, this._kNearestNeighbour);
9      Location primaryBestMatchingLocation = primaryNearestLocations[0];
10
11     _secSearchSpace = primaryBestMatchingLocation.GetLocationsWithinRadius(this.
           _radioMap, this._secSearchSpaceDist).Except(_primSearchSpace);
```

```
12
13      List<Location> secBestMatchingLocations = KNearestNeighbour(fingerprint,
            _secSearchSpace, 1);
14
15      if (secBestMatchingLocations.Count() > 0 &&
16          DistanceToFingerprintInRadiomap(fingerprint, secBestMatchingLocations[0].
                Fingerprint) <
17          DistanceToFingerprintInRadiomap(fingerprint, primaryBestMatchingLocation.
                Fingerprint))
18      {
19          this._secSpaceHistory.AddLocationToHistory(secBestMatchingLocations[0]);
20
21          if (this._secSpaceHistory.IsConsecutiveNumberOfLocsReached())
22          {
23              this._locationsCloseToEstimate = new List<Location> {
                    secBestMatchingLocations[0] };
24
25              this._bestMatchingLocation = secBestMatchingLocations[0];
26              this._estimatedLocation = secBestMatchingLocations[0].Coordinate;
27              this._secSpaceHistory.ClearHistory();
28              return;
29          }
30      }
31      else
32      {
33          this._secSpaceHistory.ClearHistory();
34      }
35
36      this._locationsCloseToEstimate = primaryNearestLocations;
37
38      this._bestMatchingLocation = primaryBestMatchingLocation;
39      this._estimatedLocation = MergeLocations(primaryNearestLocations);
40  }
```

Listing 17.1: The *ConductStrategy* method implementing the Weighted Graphs fingerprinting technique.

Initially, when calling the method, a check is performed in lines 3-6 to determine whether or not it is the first time the method is being called. In case it is, *_bestMatchingLocation* is set to null. This is necessary due to the problem that when the mobile application is initialised, it is not known where the user is located. For instance, there may be multiple entrances to the building in question or the user might choose to start the mobile application somewhere inside the building. Given that the method is called the first time, it is therefore necessary to omit conducting the actual Weighted Graphs fingerprint technique in which only a subset of the fingerprints of the radio map are subject for further calculations. Instead, the entire radio map must be examined which is done by initialising the *_primSearchSpace* variable to all the locations of the radio map. Otherwise, given that the method has been called more than once, the primary search space is initialised to the feasible locations of the

current location to which the user was previously considered to best be collocated with. The feasible locations comprise all locations recursively being neighbours to the current location within the primary search space distance.

The primary search space distance is expressed as the walking speed of the user multiplied by the update interval. As mentioned previously, the aimed update interval of BlueCAML is four seconds. Since the requirements state that the average walking speed of the users is 1.4 m/s (5 km/h), the primary search space radius is $1.4\frac{m}{s} \cdot 4s = 5.6m$.

When the primary search space has been initialised, the next step in the Weighted Graphs fingerprinting technique, conducted in line 8, is to perform the KNN algorithm to receive the K locations having fingerprints with the shortest distance to the online fingerprint *fingerprint*.

The *KNearestNeigbour* algorithm is inherited from the `NaiveFingerprintTechnique` and implements a penalty system when calculating the Manhattan distances between corresponding RSSI measurements from signaltransmitters in the online fingerprint and radio map, respectively. The penalty system punishes the Manhattan distance calculation if the online fingerprint contains RSSI measurements from signaltransmitters that are not present in the corresponding fingerprint in the radio map. The penalty encompasses adding the missing RSSI measurement from the radio map with 127 which is the highest possible RSSI measurement. This way, lower RSSI measurements result in lower penalties and vice-versa.

The location with a corresponding fingerprint with the shortest Manhattan distance, is then subsequently used for calculating the radius nodes surrounded by the user. All radius nodes, including some of the nodes that were deemed a part of the primary search space are found by calling *GetRadiusNodes* which essentially searches the entire radio map for locations having an Manhattan distance within the secondary search space radius. This radius is expressed as the walking speed multiplied by the update interval multiplied by the history size. In BlueCAML, a history size of two is used which is in accordance with a similar approach conducted by Hansen and Thomsen (2009). This yields a secondary search space with radius $1.4\frac{m}{s} \cdot 4s \cdot 2 = 11.2m$. To filtrate the nodes of the primary search space from the radius nodes, a call is made to *CalcSecSearchSpace* which returns the actual secondary search space.

From the secondary search space, the best matching location is determined by calling the NN algorithm in line 13. Notice that a check is needed to ensure that the list of locations in the secondary search space is nonempty which is the case the first time *ConductStrategy* is called. This is where the need of the history becomes evident. The identified location of the secondary search space is added to the history given that it has an Manhattan distance that is shorter than the one identified from the primary search space. This is an indicator that the system should make an illogical jump to the location from the secondary search space. However, the jump is only made if the history contains the required number of consecutive locations to the secondary search space. This check is conducted in line 21. If the illogical jump was deemed inappropriate, the K nearest neighbours found in the primary search space are used for calculating a single location to which the user is considered to be located. This functionality is done by averaging the coordinates of the K nearest neighbours. Besides

estimating this location, the fingerprints from the K nearest neighbours are saved. This is necessary in a subsequent step of the *UpdateLocation* method previously introduced, because the signaltransmitters comprising these fingerprints are candidates for being connected to as mentioned in the introduction of this chapter.

17.2 Radio Map Maintenance

The process of maintaining the radio map for a specific location involves multiple interactions between the user and the `MaintenanceEngine` controlled by the `Presenter` class. Figure 17.3 shows the sequence of calls under normal operation and the different classes involved in this task, and how they interact with each other. Logic involved in cancellations and errors have been omitted for verbosity. Furthermore, the interaction from the `Presenter` to the View Layer has been simplified by merging multiple calls into the *NotifyUser* call. This call abstracts a number of calls which change the View Layer either by showing notification messages or changing the available menus.

Notice that the `Presenter` calls *DoNextMaintenanceAction* on itself, which is a function executing the necessary steps in order to accomplish this action. The actions are, as described in Section 14.2.2, find signaltransmitters, conduct fingerprint when facing north, east, south, and west, and upload the fingerprint to the back-end. The four measurements are conducted after prompting the user to confirm her direction and the View Layer calls back to the `Presenter`, which can be seen at the beginning of the loop. Note that the `ConnectionManager` class further utilises the `BluetoothDeviceWM6` class which further uses the `BluetoothSocket` and `BluetoothSocketEndPoint` classes. However, these are not depicted due to verbosity.

In the following, a more detailed description is given for the transmission of a fingerprint to the back-end and interaction between the `MaintenanceEngine` and the `Presenter`. This logic is part of the `BluetoothDeviceWM6` class.

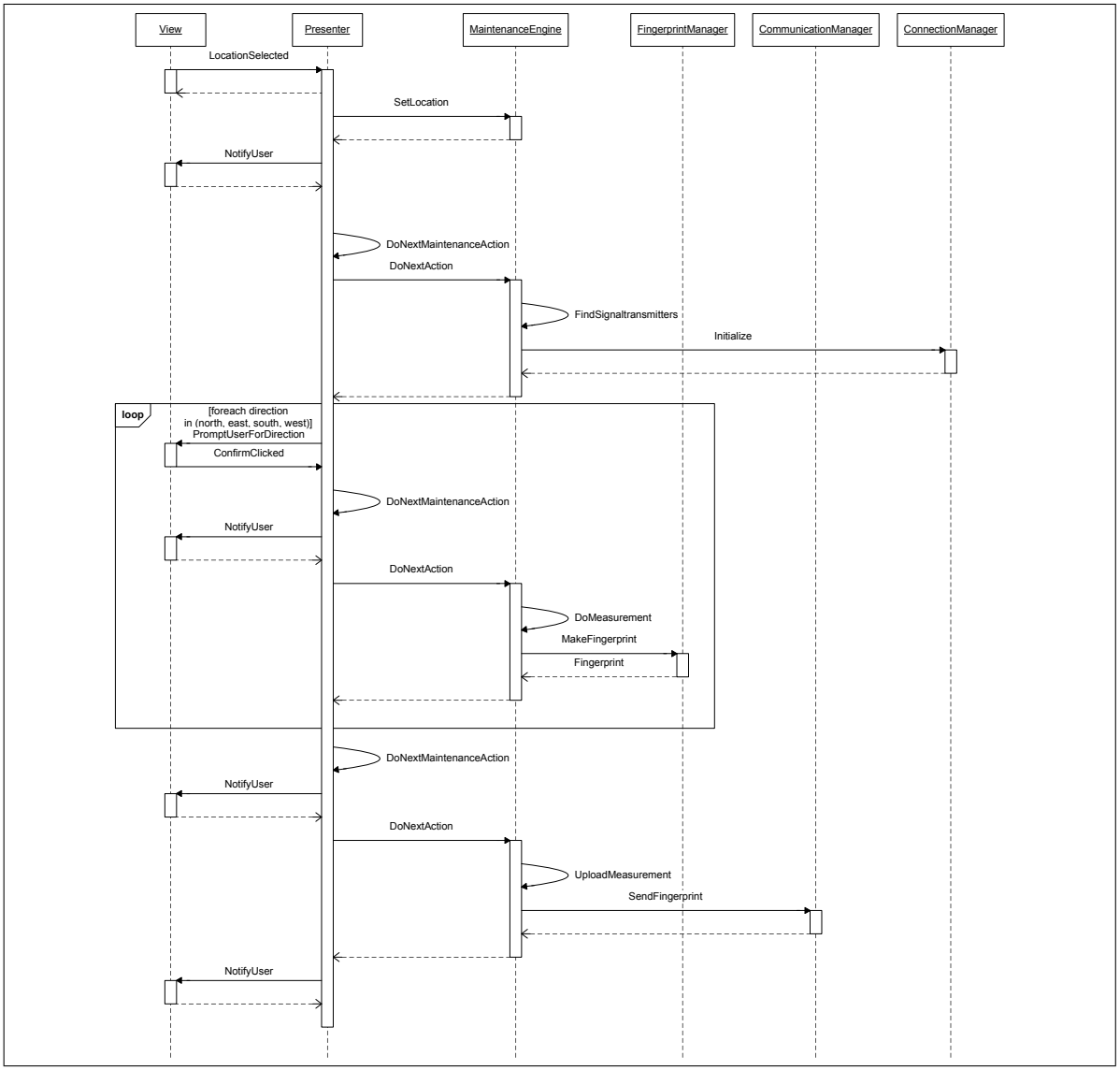


Figure 17.3: UML sequence diagram depicting the sequence of calls involved in maintaining the radio map at a specific location.

17.2.1 Transmission of Fingerprint

In order to update the radio map in the back-end, the fingerprint is first transmitted to a smart signaltransmitter, which in turn is responsible for transmitting it further to the back-end.

The code example in Listing 17.2 shows the *TransmitFingerprint* method responsible for transmitting fingerprints to a signaltransmitter.

```
1  public void TransmitFingerprint(Fingerprint fp, byte[] mac, int locationId)
2  {
3      Socket socket = BluetoothSocket.GetLocalSocket();
4      socket.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReceiveTimeout,
           1000);
5      socket.Connect(new BluetoothSocketEndPoint(mac, SERVER_PORT));
6
7      string fpXmlString = fp.ToXmlString(locationId);
8
9      DataPacket request = new DataPacket();
10
11     request.Command = 'S';
12     request.Status = 0;
13     request.Payload = Encoding.UTF8.GetBytes(fpXmlString.ToCharArray(), 0, fpXmlString.
           Length);
14     request.PayloadSize = request.Payload.Count();
15
16     this.SendPacket(socket, request);
17
18     DataPacket response = this.ReadPacket(socket);
19
20     socket.Close();
21
22     if (response.Status != 0)
23         throw new ServerCommunicationException();
24 }
```

Listing 17.2: The *TransmitFingerprint* method is responsible for sending the fingerprint through a RFCOMM socket to a signaltransmitter.

The *TransmitFingerprint* method takes three parameters: the `Fingerprint` to transmit, the MAC-address of the destination signaltransmitter, and the id of the location where the fingerprint was made.

An RFCOMM socket connection is established in line 3-4 using the `BluetoothSocket` helper class. Then, in line 6-13 a `DataPacket` is prepared, setting the command and payload. The command "S" instructs the signaltransmitter to receive a new fingerprint, which is located in the payload part of the packet.

In line 15, this request packet is sent, and a response packet is read in line 17. The status of the response is checked to see if the fingerprint was correctly received by the signaltransmitter, which is the case when the status is zero. If not, an exception is thrown. In

general, this procedure can throw two exceptions: `ServerCommunicationException` if some error occurred on the signaltransmitter, and `SocketException` if the connection attempt to the signaltransmitter failed or made a time out.

17.2.2 Presenter and Maintenance Engine Interaction

The interaction between the `Presenter`, View Layer, and the `MaintenanceEngine` is complicated by the introduction of multiple threads. Listing 17.3 shows an excerpt of the *`DoNextMaintenanceAction`* method in the `Presenter` class which illustrates how this is conducted.

```
1  public void DoNextMaintenanceAction()
2
3      ...
4
5      case MaintenanceEngineActions.MeasureNorth:
6      case MaintenanceEngineActions.MeasureEast:
7      case MaintenanceEngineActions.MeasureSouth:
8      case MaintenanceEngineActions.MeasureWest:
9          this._view.ShowNotice(MSG_COLLECTING_INFO);
10
11         ThreadPool.QueueUserWorkItem(new WaitCallback(delegate(object state)
12         {
13             this._maintenanceEngine.DoNextAction();
14
15             this._view.BeginInvoke(new MethodInvoker(delegate()
16             {
17                 if (this._maintenanceEngine.NextAction == MaintenanceEngineActions.
                       UploadMeasurement)
18                     this.DoNextMaintenanceAction();
19                 else
20                     this.PromptUserForMaintenanceDirection();
21             }));
22         }));
23
24         break;
25
26     ...
27
28 }
```

Listing 17.3: Excerpt of the *`DoNextMaintenanceAction`* method in the `Presenter` responsible for measuring values from four different directions.

As shown, the excerpt is based on a switch statement which checks on the next action of the `MaintenanceEngine`. This part handles the case of the next action being a measurement in one of four directions.

In line 11, a job is enqueued to be executed by a worker thread in the thread-pool. This is done in order to allow the UI thread, which is executing this action, to proceed with other tasks such that the user interface can continue to be responsive. On line 13, the `MaintenanceEngine` is called in order to execute its next action, which in this case involves constructing a fingerprint.

In line 15, the *`BeginInvoke`* method is called on the `View` class which requests the delegate, defined in line 12-22 to be executed on the UI thread when possible. This is done to exit the worker thread and return to the UI thread. In line 17, the new next action of the `MaintenanceEngine` is then checked, and if the next action is *`UploadMeasurement`*, it calls the *`DoNextMaintenanceAction`* method on itself again in order to get into the case statement which handles this action. If not, the next action involves another directional measurement, and thus the *`PromptUserForMaintenanceDirection`* method is called which interacts with the View Layer in order to get the user to change direction.

17.3 PIFC Renderer

The PIFC renderer is a reused package which has been changed slightly in order to better integrate with the remaining application. The main change made to the PIFC renderer is the movement of a private Form object to the argument list of it. Previously, the lowest layer of the renderer created its own Form for creating a window visible to the user, which it added buttons and other logic to. The problem is that this Form is only accessible from the renderer itself. This is changed such that a Form is created externally, and given to the renderer which then is responsible for rendering the map on the Form.

Furthermore, the PIFC Renderer works with a number of objects which can be placed on the map such as Wi-Fi hotspots. These are displayed on the map and support the user clicking on them. In order to show maintainable locations on the map, a new type of these objects are added and necessary APIs created such that they can be added and removed at runtime. This is necessary to do directly in the PIFC Renderer, since it does not support adding other types of objects to the rendering processes than a predefined set.

Besides these structural changes, only a few minor corrections have been made to better support changing the rendered map from the code controlling it. As an example, the `PIFCRendererFacade` class, which acts as an interface between the PIFC Renderer and the View, is modified such that zoom and panning of the map can be controlled through it.

18 Testing

Throughout the development of BlueCAML a number of different test methods have been applied in order to improve and check the overall quality of BlueCAML.

The following tasks are documented in this chapter.

- The memory usage of the signaltransmitter application has been checked for no leakage of memory under operation.
- The power consumption of the mobile application is examined.
- Unit tests have been conducted for the program in order to reduce faults and ensure that it follows the specification.
- A usability test has been conducted to improve the user interface and make it more accessible for users.
- The accuracy of the positioning system has been tested in practice in an environment inside a building.

18.1 Resource Usage

Different approaches exist for measuring resource usage in software. More modern approaches consist of static code analysis where the source code is analysed by a program which estimates the upper bounds on resource usage (Heckmann and Ferdinand, 2005). In addition, there also exist tools for dynamically examining the resource consumption of the particular application.

C++ used for the development of the signaltransmitter application, does not include a garbage collector. Hence, dynamically allocated memory on the heap must manually be deallocated to reduce the memory footprint in terms of introduced memory leaks during the lifespan of the application. This is especially of importance with regards to the signaltransmitter because it contains relatively low resources and therefore can be expected to be prone to memory leaks. Manually handling memory allocation and deallocation is often subject to faults in the program (Evans, 1996).

For analysing the memory consumption and determine whether or not there have been introduced memory leaks, one of the Valgrind (Valgrind, 2010) tools has been applied to the signaltransmitter application. Valgrind is a set of tools that can be used for memory debugging and profiling among others. The Memchecker tool of Valgrind is capable of replacing the standard C memory allocator with its own implementation which is the essential component in detecting memory leaks and other memory related faults such as reading memory which has not yet been allocated.

In the following, the test results from using Valgrind on the signaltransmitter application are shown. The test has been conducted on the interface it exposes to the only communication instantiator; the mobile device. This means that Valgrind has been used both when the mobile device requests the radio map, and when it uploads a fingerprint. Due to both interface functions having the same memory footprint, the table shown in 18.1 contains only a single row.

Definitely lost	Indirectly lost	Possibly lost
0 bytes	0 bytes	144 bytes

Table 18.1: Memory leaks in the signaltransmitter application.

As shown, the signaltransmitter application does not introduce memory leaks in the categories "definitely" and "indirectly lost". These are the most crucial memory leaks. Valgrind reports memory leaks as "possibly lost" when there is a high likelihood of introducing a memory leak. However, given that Valgrind executes an application with debugging information, it is capable of pointing out precisely what caused the memory leak. With respect to the 144 bytes that possibly are lost, Valgrind reports that this is due to the thread continuously processing the HTTP request queue. This memory leak can be considered as a false positive since the memory leak is only introduced if the program halts because the thread is not destroyed properly. However, in practice this will never be the case since the thread will be continuously running in an endless loop and does not at any point during program execution exit. For test purposes, the loop, however, needs to be broken after a single iteration since otherwise it would not be possible for the application to terminate.

18.2 Power Consumption

When developing mobile applications, power consumption is a concern that should be taken into account. Therefore, the power consumption of the mobile application is examined to ensure it does not drain the battery after short usage. To do this, the Windows Mobile Power Management (Johnson, 2009) benchmarking tool is used. This tool measures the power consumption over time, meaning that it can run simultaneously with the mobile application continuously estimating positions.

Figure 18.1 depicts the outcome of the Windows Mobile Power Management tool.

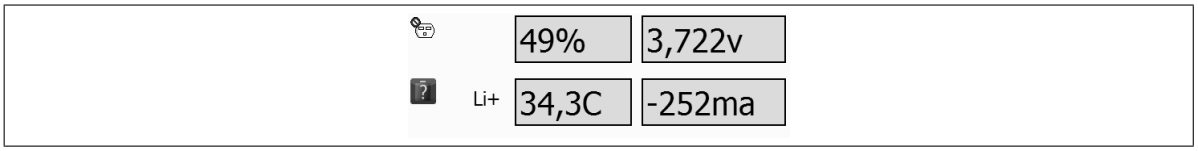


Figure 18.1: Power consumption of the mobile application while conducting position estimates.

As shown, the power consumption is 252mA. From the HTC Diamond specification (HTC, 2010), it is stated that the battery capacity is 900mAh. Therefore, it is estimated that the application can be run for $\frac{900mAh}{252mA} = 3.6h$ before the battery is drained. This is considered sufficient since it is assumed that a user can recharge her mobile device once a day and that the application is not assessed to be used that long each day.

18.3 Unit Testing

In Section 8.1, it is decided that unit, integration, system, regression, and acceptance tests are conducted. In this chapter, however, only a concrete example of a unit test with equivalence partitioning is shown. The reason for not providing examples of the remaining test types is due to verbosity.

As an example of unit tests using equivalence partitioning three tests for the `ConnectionManager` class are shown in this section. Specifically, the *`UpdateConnectedSignalTransmitters`*, which is responsible for updating a list of currently connected signaltransmitters, is tested.

This function uses five values; four of them given through the constructor of the `ConnectionManager` class and one given as an argument to the tested method. The values given to the constructor set the minimum and maximum number of connected signaltransmitters, stores a reference to the Bluetooth device, and sets the maximum amount of time allowed for connecting to new signaltransmitters. To the method, a list of the last measured fingerprints is given.

In the following example, the maximum amount of time the function is allowed to connect to the signaltransmitters is changed in order to test the different behaviours of the method. The specification states the following.

1. If possible, the connection manager must connect to the minimum number of connectable signaltransmitters regardless of given time limits.
2. If additional time is available, then connect to an additional signaltransmitters.
3. The connection manager must not connect to more than the maximum allowed signaltransmitters regardless of remaining time for initiating connections.

From the above, the connection time limit can be partitioned into three equivalence classes:

1. Connection time limits being below the required time for connecting to other than the minimum number of signaltransmitters.
2. Connection time limits being above the required time for connecting to more than the minimum but not the maximum number of signaltransmitters.
3. Connection time limits being above the required time for connecting to more than the maximum number of signaltransmitters.

In each case, a test can be made which sets this time limit, calls the method, and checks the resulting number of connections. A test which tests each of the defined cases is given in Listing 18.1. The test is constructed using the NUnit test-framework which is chosen because of its ability to be easily integrated into Visual Studio.

```
1  [Test]
2  public void TestExtendedButLimitedConnection()
3  {
4      ConnectionManager connectionManager = null;
5      Fingerprint fp = GetFingerprintWithMeasurements(5);
6
7      BluetoothDeviceDummy bluetoothDevice = new BluetoothDeviceDummy();
8      bluetoothDevice.DummyConnectionTimeout = 200;
9
10     // case 1 - Only time to connect to minimum
11     connectionManager = new ConnectionManager(2, 4, bluetoothDevice, 100);
12     connectionManager.UpdateConnectedSignalTransmitters(new List<Fingerprint> { fp });
13
14     Assert.AreEqual(2, connectionManager.ConnectedSignalTransmitters.Count());
15
16     // case 2 - Connect to more than minimum but not maximum
17     connectionManager = new ConnectionManager(2, 4, bluetoothDevice, 500);
18     connectionManager.UpdateConnectedSignalTransmitters(new List<Fingerprint> { fp });
19
20     Assert.AreEqual(3, connectionManager.ConnectedSignalTransmitters.Count());
21
22     // case 3 - Connect to max signaltransmitters
23     connectionManager = new ConnectionManager(2, 4, bluetoothDevice, 5000);
24     connectionManager.UpdateConnectedSignalTransmitters(new List<Fingerprint> { fp });
25
26     Assert.AreEqual(4, connectionManager.ConnectedSignalTransmitters.Count());
27 }
```

Listing 18.1: Example of a unit test using equivalence partitioning for the `ConnectionManager` class.

Line 5 prepares a fingerprint for being input to the tested method. This fingerprint contains measurements to three signaltransmitters the connection manager can connect to.

In line 7 a Bluetooth device dummy is instantiated. Normally, the Bluetooth device communicates with the hardware, but in order to isolate the connection manager, this dummy

is used. The dummy is constructed such that it allows the connection manager to connect to all signaltransmitters. The dummy is further configured in line 8 in which its connection time is set to 200 milliseconds. This is done to control the number of possible connections that can be made.

In line 11, a connection manager is instantiated such that it requires minimum two and maximum four connections. Furthermore, the connection time limit is set to 100 milliseconds. By setting this to 100 milliseconds, there will not be enough time to connect to the minimum amount of connected signaltransmitters, which takes approximately 400 milliseconds. The tested method is called in line 12, and the number of connections is checked in line 14.

This test is repeated for the two other cases, by changing the connection time in order to test the equivalence classes. In line 16-20, case 2 is tested and in line 22-26 case 3 is tested.

Note, that in relation to this test, the outcome is influenced by time. Due to the application and underlying system not having real-time properties, it cannot be ensured that the test will succeed regardless of the environmental state. For example, in test 2, if, for some reason, the first two connections exceed 500 milliseconds of time usage, this test will fail.

18.4 Usability Testing

Emphasis must be put in usability testing due to the importance of users being able to understand and use BlueCAML, and thus it must fulfil the usability quality criteria. On a previous project, the informal usability testing approach described by Krug (2005) was applied successfully. The approach resulted in a number of critical usability problems being discovered. These were corrected and resulted in improving the application from a user-perspective. Due to this success, a similar approach is conducted in this project.

Since usability primarily is improved for the maintenance of the radio maps, this part will be tested in the usability test. Therefore, different tasks covering the aspects of this are made and asked solved by the test users.

Two students attending the 8^{th} semester in Software Engineering are used to test the system. The purpose of BlueCAML is described to the test users, and afterwards, a set of tasks are given which must be solved one by one. Appendix B describes the introduction and the tasks.

A screenshot of the final version of the mobile application is shown in Figure 18.2. This figure can be used as reference in the following.

18.4.1 Results from the First Test

The first test is conducted and results in a number of observed problems. These problems are then corrected before conducting the next usability test, which is, based on experience, yields better results. This way, it is possible to test the changes made to the problems uncovered in the first test, and possibly uncover new problems now introduced.

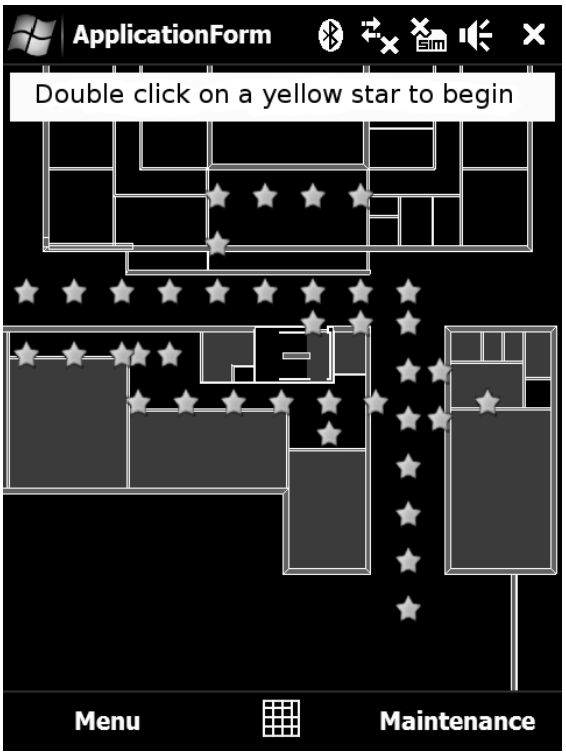


Figure 18.2: Screenshot of the final version of the mobile application in maintenance mode.

In the first test the following problems are observed.

1. The user has difficulties in seeing the current estimated position shown on the map before entering maintenance mode.

2. The user comments on the Wi-Fi hotspot icons which are shown on the map, and asks why they are present and whether they are of any importance to the usage of the system.

3. The user does not know if the "update radio map" or "change mode" buttons should be used for entering maintenance mode. The user thinks that the button "update radio map" should be used for transmitting corrections of the radio map and not download an updated radio map.

4. Yellow stars are shown to the user when entering maintenance mode, but what these represent are not initially apparent.

5. After explaining what the yellow stars represent, it is still not apparent to the user how to interact with these in order to maintain a given location. Specifically, the need for double clicking the yellow stars is difficult to the user.

6. The user does not know how to react when instructed to change direction to north, south, etc. The user wants to consult a real compass in order to find the desired direction in which the measurement is to be conducted.

The following changes are made in order to correct the above.

- The Wi-Fi hotspots are removed from the map in order to correct item 2.
- The "Update radio map" menu item is renamed to "Fetch new radio map", in order to remove some confusion in regards to this menu item in item 3 and 5.
- The "Change mode changed" menu item is renamed to "Maintenance mode" and "Positioning mode" depending on the current mode, such that it is more apparent that this menu item is used to change to the desired mode, which was difficult in 3.
- Instructions are added to the screen informing the user that the starts should be double clicked when in maintenance mode. This should help remove confusion in item 4 and 6.

No solutions are implemented for item 1 and 6 before the next usability test is conducted as a result of time scheduling. However, these are corrected afterwards.

18.4.2 Results from the Second Test

In the second test, the following observations are made in regards to usability problems with the application.

1. The user mentions problems with the visibility of the currently estimated position on the screen and mentions that the position marker is very small.
2. The user is confused in regards to some rooms not being rendered in the same colour on the map as other rooms and what this could possibly mean.
3. The user has difficulty in seeing the stars when entering maintenance mode since the map is zoomed out and thus the stars are very small.
4. The user has difficulty in selecting a location by double clicking a star. Initially, the user only click once on the star and determines that this is not the correct way to interact with them.
5. The user exhibits problems when asked to face a specific direction. As in the previous test, the user cannot figure out how to face the specific directions when not given tools showing where north, south etc. are.

From this test, it is concluded, that the previously resolved issues are not observed again. Specifically, the problem of changing from positioning to maintenance mode is more intuitive than before. The two problems which were not fixed in the first test are observed again.

The following changes are made to the application based on the usability test:

- The current position marker is increased in size with a factor two in order to fix item 1.
- Arrows are added in order to show the user which direction she should face in order to fix item 5. Thus, if the user should face the north wall of the building, an arrow now shows the direction the user needs to face in relation to the map.

  Other possible solutions could be to add a compass to the interface, but this requires support from the hardware.

Item 2 is not fixed since this is a problem with the provided PIFC model and missing room information in the file which result in the PIFC Renderer rendering the rooms without a floor.

Item 3 is not fixed because of limitations in the PIFC Renderer, which does not allow changes to the map and zooming in one operation, and it does not provide the necessary functionality to query these operations in a straight-forward manner.

The task of double clicking, as described by item 4, is not easy, and only requiring a single click would make it easy to activate the wrong location by mistake. A solution would be to click once to mark a position and then press another button to confirm in order to verify the selection. Implementing this solution would require changing the PIFC Renderer which uses double clicking to respond to single clicks. If implemented, the usage of the PIFC Renderer would therefore be inconsistent since double clicking should be used in some situations and not in others. Therefore, the proposed solution is not implemented.

18.5 Accuracy and Precision of Position Estimates

In the following, the accuracy and precision of the position estimates are determined on the basis of a concrete test scenario. First, the approaches for collecting the offline and online RSSI measurements are described.

18.5.1 Offline Stage

The fingerprints for the offline stage are made as described in Chapter 7. This means that fingerprints are made in intervals of three meters, where RSSI values for each detectable signaltransmitter are measured for 2000 milliseconds as is determined in Chapter 7 for the four orientations: north, south, east, and west.

Figure 18.3 depicts the area used for positioning. The area consists of both rooms and corridors. Also, the area contains a parallel path, which is assumed not to be clearly distinguishable when calculating the position estimates. Therefore, the area represents a varying and challenging environment which creates the foundation for testing how well position estimation by BlueCAML is in general.

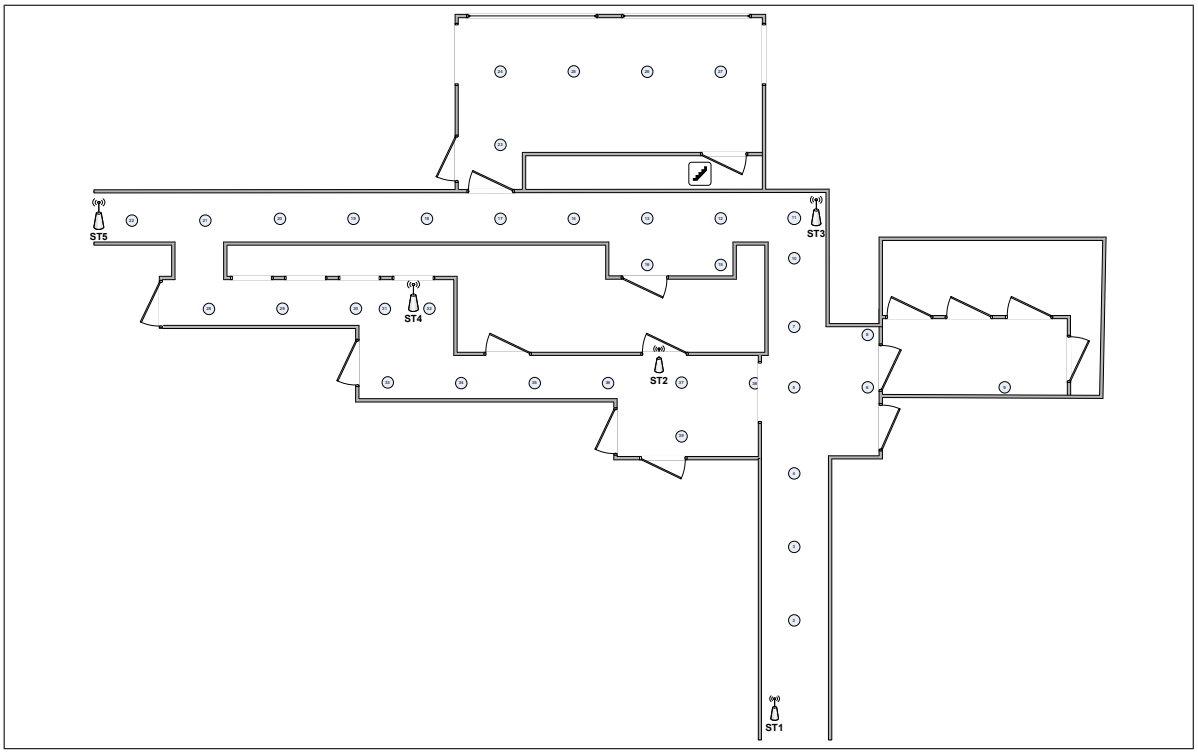


Figure 18.3: The test scenario which forms the basis for determining the accuracy of BlueCAML. Grey circles represent locations at which fingerprints have been conducted during the offline stage.

The signaltransmitters are strategically placed in the area, such that the minimum detectable signaltransmitters on each fingerprint is three, which is assumed to be sufficient for position estimates.

18.5.2 Online Stage

In the online stage, a walking path covering the majority of the area is chosen as depicted in Figure 18.4. The space between two consecutive black circles indicates the path in which a fingerprint is collected. This means that the person collecting the fingerprints in the online stage is constantly moving along the path. In the authors' opinion, this gives a more fair picture of the accuracy than if the person had been statically placed at a location at which the fingerprint was collected. To create a large data set, the route shown in the figure, has been traversed three times.

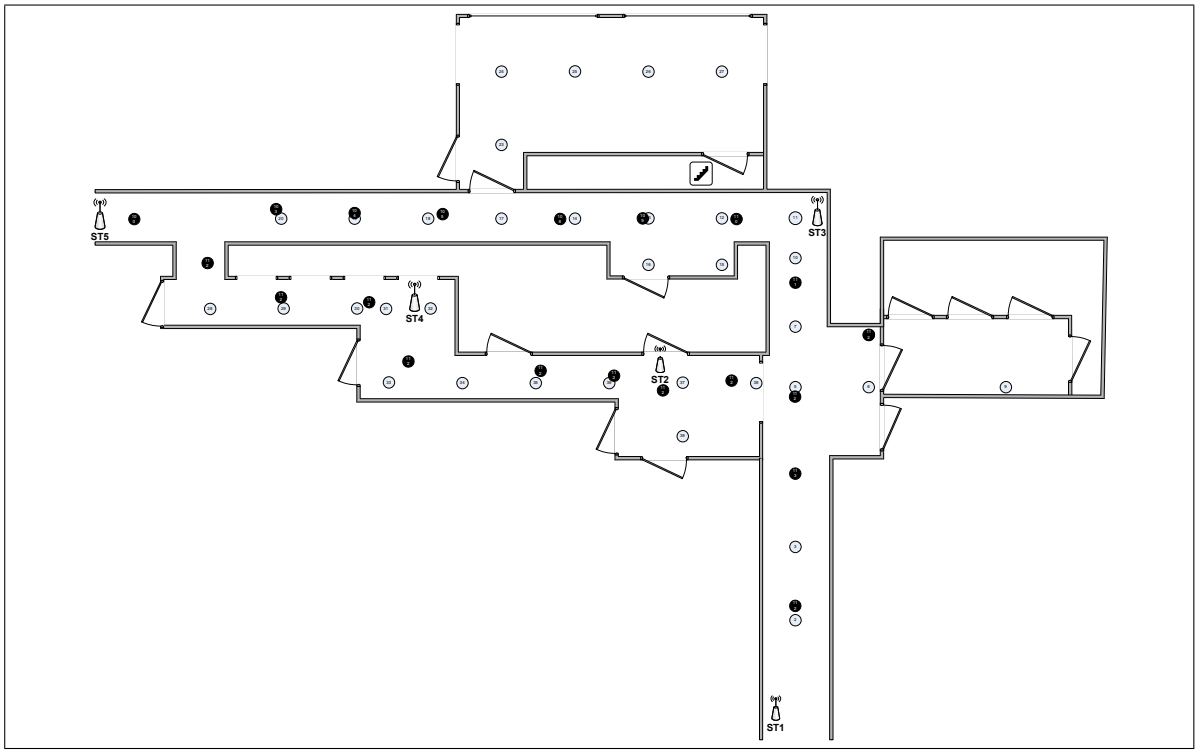


Figure 18.4: The traversed route in the test scenario. Grey and black circles represent locations at which fingerprints have been conducted during the offline and online stage, respectively.

18.5.3 Calculation of Accuracy

The evaluation of accuracy and precision of BlueCAML takes starting point in the requirement of three meters accuracy with 80% precision. The accuracy at 80% precision is used as the primary criterion for evaluating the overall accuracy of BlueCAML, because the main objective is to assure the users of BlueCAML how much an estimated position deviates in most cases and not the relatively high accuracy seldom achievable.

For determining the accuracy and precision achievable by BlueCAML, a benchmarker tool is developed. The incentive for developing this tool is that different techniques with various configurations, such as varying K in KNN, can be compared. A short description and screenshots of the benchmarker tool can be found in Appendix C.

As previously described in Chapter 5, KNN combined with the Weighted Graphs technique is used in BlueCAML. However, to evaluate Weighted Graphs' effect, the accuracy calculations are done both with and without Weighted Graphs. Furthermore, K in KNN is varied in order to show how this constant affects the result.

The accuracy is determined by calculating the difference between the estimated positions and the actual positions. By grouping the observed errors for each meter, it is possible to determine the precision.

Figure 18.5 shows the accuracy of the KNN strategy without Weighted Graphs with K ranging from 1 to 4.

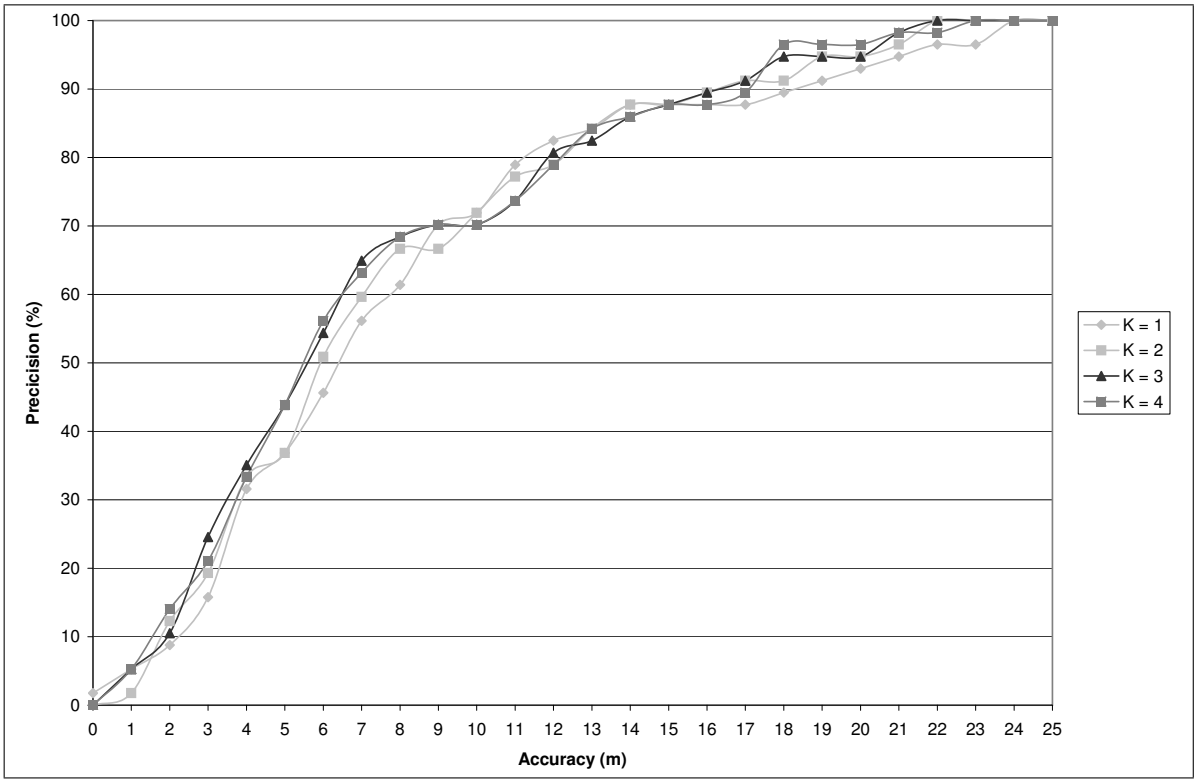


Figure 18.5: Accuracy and precision obtained using different K in KNN.

As shown, the results with different K in KNN do not vary significantly. The most distinct part is observed in accuracies between three and nine meters, where the result with $K = 3$ approximately has a precision 5% points above that when $K = 2$. In the 80% to 100% precision interval, which, as previously described, is of interest in respect to BlueCAML, $K = 3$ and $K = 4$ slightly outperform the other two. Because it cannot be determined which of the two yields the better result, $K = 3$ is chosen as most appropriate since it requires minor less computations.

Generally, it is observed for all configurations, that the estimated positions get stuck when the user walks through the curve in the upper right corner. This means that position estimates after this point generally have a low accuracy. Even updating the radio map in the specific corner does not resolve the issues, and hence it must be a problem with the offline fingerprints in that area being too close to the online fingerprints in other areas.

The same approach is used with KNN combined Weighted Graphs. Figure 18.6 shows the results using this search strategy while varying K between 1 and 4.

As in the case of using KNN without Weighted Graphs, the results with Weighted Graphs show the same tendency, that is, varying K does not provide a significant change in accuracy

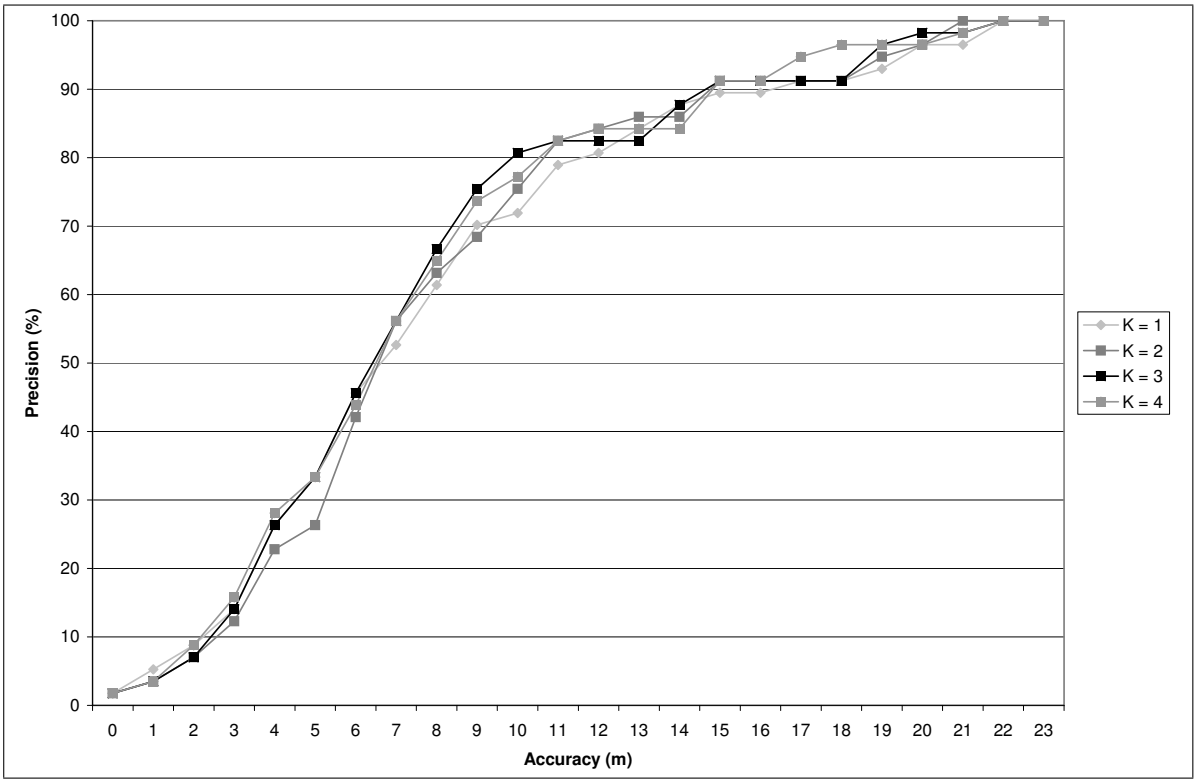


Figure 18.6: Accuracy and precision obtained using different K in KNN together with Weighted Graphs.

nor in precision. These results might be caused by similar reasons as in the test with KNN without Weighted Graphs. $K = 3$ is chosen as the most appropriate value for similar reasons as in the previous test.

In order to determine whether Weighted Graphs improves the results from KNN, the results from both of these are compared. K is set to 3 based on the conclusion made in the previous test.

Figure 18.7 shows how these two strategies compare.

The results of comparing KNN with and without Weighted Graphs show that there is not a significant improvement in using KNN with Weighted Graphs. At the desired precision of 80%, Weighted Graphs yield better results but besides, KNN tends to be better. However, during simulation with the benchmarker tool, it is discovered that KNN tends to randomly select locations in the scenario. Therefore, an estimate based on these may in average yield good results even though the estimated positions are randomly determined. The Weighted Graphs technique is more likely to follow the path of the user, but due to the previously mentioned problem of getting stuck, the subsequent estimates are relatively poor.

The conclusion is, that by using Weighted Graphs, an accuracy of ten meters can be obtained with a precision of 81%.

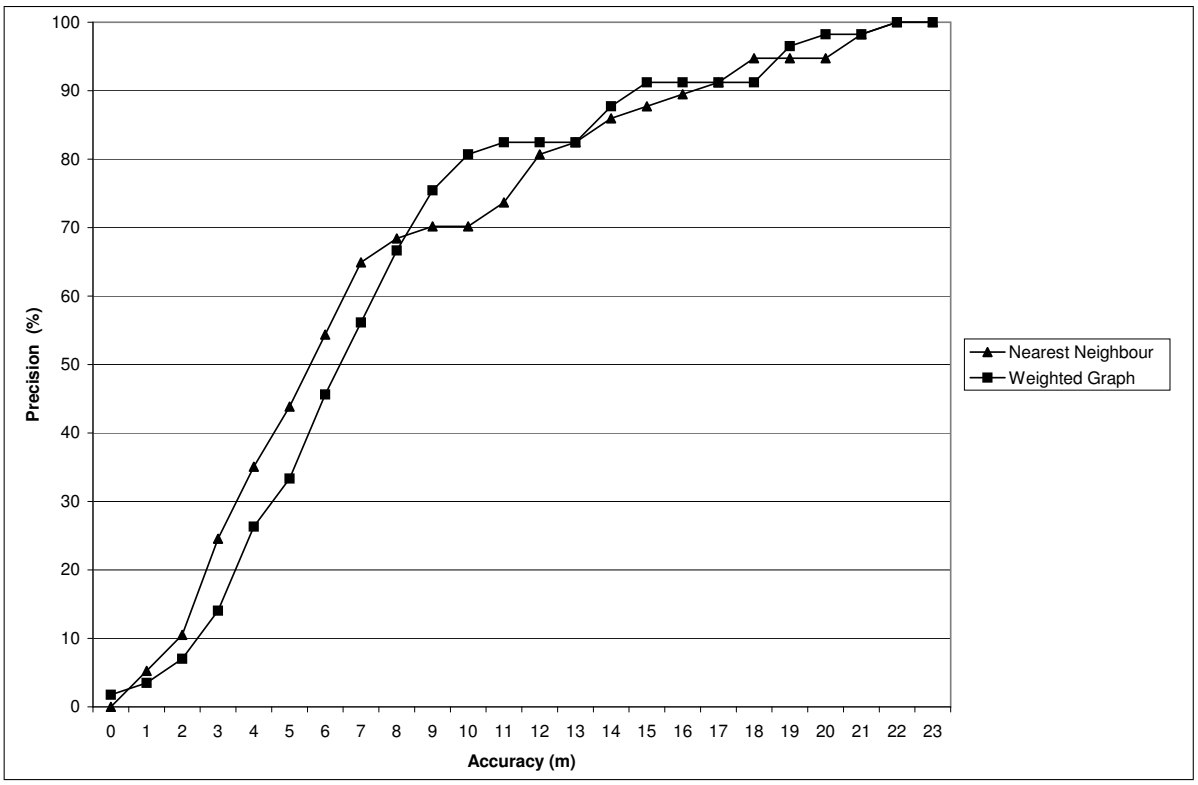


Figure 18.7: Accuracy and precision obtained from KNN and KNN with Weighted Graphs when $K = 3$.

Part V

Conclusion

19 Conclusion

In this project, focus has been put in developing BlueCAML which is a system aimed at providing position estimates of mobile devices using Bluetooth and show this to the user on a provided map of the particular building.

With regards to conducting position estimates, the fingerprint technique is used. Due to many influencing factors such as varying environmental conditions, the accuracy and precision of the fingerprint technique degrades over time, hence, estimating erroneous positions of the user. Because of this problem, a significant effort has been put in introducing a method allowing the users to perform corrections to the system thereby targeting the accuracy and precision levels remain stable regardless of influencing factors. This also adds a distributed perspective of the system due to maintenance being put on the user side. Therefore, the development of BlueCAML complies with the overall theme of this project, namely, *Distributed and Mobile Software*. The entire development of BlueCAML is in accordance with the learning goals from the study regulation, namely, *Demonstrate knowledge and understanding in analysing, designing, implementing, and evaluating software for a mobile platform* and *Consider concepts and opportunities within mobile technologies*.

As described, the accuracy of BlueCAML is ten meters with a precision of 81%. This does not fulfil the requirement of having an accuracy of three meters with that particular precision. However, it still opens for the possibility of further research in Bluetooth for indoor positioning systems, since a number of other fingerprinting techniques that might enhance the accuracy can be applied. Even if these do not enhance the accuracy, the current accuracy is more than enough for systems aiming at room-level position estimation.

BlueCAML is based on an architecture involving three different components: mobile device, signaltransmitter, and back-end. The back-end stores information relevant to the buildings participating in BlueCAML. In addition, the back-end is responsible for retrieving the necessary information needed for the fingerprint technique to users on demand. In this regard, it also implements functionality for processing and storing the corrections made by the users. Signaltransmitters are strategically placed around in the building and act as proxies between the mobile devices and back-end exposing an API which the mobile devices can use both in regards to maintaining the radio map and for retrieving it as well.

In the present effort, the realisation of BlueCAML complies with all requirements initially set except for the time constraint between consecutive calculated position estimates. The goal was to calculate the position estimate every four seconds, however, this requirement cannot be upheld since it takes 5.8 seconds if a single connection is established. This result is primarily due to the time needed for establishing a connection for RSSI measurements and the actual measuring of RSSI values. However, given the assumption that a new connection is not needed between consecutive position estimates, the time required is reduced to 3.1 seconds.

Due to the requirement of an API on the mobile device allowing the retrieval of RSSI values of the local Bluetooth adapter, the choice of platform is unfortunately deemed limited. Only the Windows Mobile platform exposes this functionality as of this writing which results in a limited target group due to Windows Mobile currently having a relatively low market share on the mobile market. It would have been beneficial to use Symbian or even target, to some extent, platform independence with Java.

In addition, an important discovery has been made. Specifically, being the first actual mobile application developed by the authors, it has been discovered that one should to a more wider extent know what functionality is required to be available for instance in terms of APIs than for similar development for a desktop application where this functionality often is taken for granted. For instance, it was discovered that a ping functionality was vital for ensuring correct measurements as a consequence of the implementation of RSSI values and additional handling of these. The ping request was a necessity for accommodating an important discovery made in relation to how the Bluetooth hardware and Bluetooth stack implementation handle RSSI values in terms of caching. Specifically, RSSI values are not measured whenever requested from the API, but instead include caching the latest ones which are then retrieved. This is undocumented and the Bluetooth specification does not make explicit what an actual Bluetooth implementation must comply with. This is crucial in terms of positioning based on RSSI due to the reliance on new RSSI values of the current position.

The communication channel between the mobile device to the back-end via the signaltransmitter has also been subject to significant effort due the establishment of an application layer communication protocol. Besides the design, verification has been paramount to among others ensure that the protocol does not potentially result in deadlocks. In this regard, analysis has been conducted in the field of static verification and most notably in the UPPAAL model checking tool. Using model checking for this purpose is appropriate due to the easiness of modelling the states and actions of a communication protocol. Through the utilisation of this, it can be concluded that the protocol is deadlock free. Also, modelling the communication protocol helped in discovering design flaws in the communication protocol.

Testing has also been subject for emphasis. Various methods and techniques have been examined through analysis. In particular, tests have been generated through equivalence partitioning and boundary analysis and executed with NUnit. Furthermore, test adequacy has been determined through test coverage with the NCover tool. The emphasis on test and verification complies with the learning goals: *Account for and assess opportunities in verification and validation of software systems*, *Demonstrate skills in applying techniques,*

methods, models, and tools for test and verification of software systems, and *Gain knowledge in using the model checking tool UPPAAL for verification.*

Besides the actual development of BlueCAML, a comprehensive pre-analysis has been conducted in order to delve into important issues that influence fingerprinting and positioning in general. This pre-analysis has primarily focused on experimentally confirming the issues presence and in addition their amount of impact and has thereby grounded the basis for which issues BlueCAML needs to accommodate in order to be as good and innovative as possible. The experiments have given birth to several analysis tools developed by the authors for investigating the individual influencing factors, which among others comprise, how orientation of users affect RSSI measurements, density of people in the area, and the radio map becoming obsolete over time to name a few. The conclusions made in each of these experiments have been taken into account in the development of BlueCAML except how varying people density should be accommodated remains without a concrete solution.

An extensive amount of resources and research effort has been put in confirming the hypothesis that the varying amount of people can be modelled by measuring the difference in environmental changes from reference measurements to current measurements and use regression analysis. Unfortunately, the discoveries made in this regard conclude that a simple relationship does not exist and thereby leaves the question of how to address the issue open. Nonetheless, the authors still see the research effort as a scientific contribution in this area since it generally is a problem for indoor positioning systems. This analysis is in accordance with the learning goals: *Examine different positioning techniques, implement one or more techniques, and test them in a practical setting* and *Gain knowledge in the capabilities of Bluetooth in regards to indoor positioning systems.*

20 Discussion

The aim of this chapter is to comment on some of the aspects, solutions to discovered problems, and things which could have been done differently. Some of these are of concern if BlueCAML should be deployed in a real world setting and some merely improvements. Besides, the purpose is also to comment on the development related issues which should be taken into account in future projects.

Fingerprinting

As pointed out in the report, a problem of using fingerprinting is the effort required to make the offline stage and afterwards the effort required to maintain the radio maps. In some scenarios, the offline stage could be expected to be of interest for the service provider, such as shopping malls. In such cases, the problem might be relatively low. In contrast, scenarios where the owners of buildings cannot see the profit of having this service, there might be low interest in spending resources in the offline and maintenance stages. BlueCAML is primarily directed at the latter scenarios, because it provides a relatively low-cost infrastructure and it provides a means of relocating the effort required for the stages from the service provider to the users of the system.

In respect to the accuracy, BlueCAML does not achieve similar results compared to Hansen and Thomsen (2009), who achieved better results using Wi-Fi. The main difference is how often it is possible to update the user's position, and hence how far the user can walk within this interval. In Hansen and Thomsen (2009), the user is estimated to be walking three meters in the update interval, whereas in BlueCAML the user walks six meters. This means that the primary search in the Weighted Graphs technique, is twice the size in BlueCAML in respect to Hansen and Thomsen (2009), hence, an accuracy of less than six meters cannot be expected.

To accommodate for the observations of using the Weighted Graphs technique, the density of fingerprints in BlueCAML could be decreased such that it matches the walking distance between the intervals. This would decrease the primary and secondary search spaces, which might help to do more accurate estimates.

Dynamically Changing the Radio Map

The problem regarding dynamically changing the radio map in respect the state of the environment had high priority, since all present solutions to this problem seemed to require a significant effort to carry out. However, as mentioned the simpler solution proposed by the authors did not work and no other solution is implemented. Therefore, if BlueCAML is to be set into production, this problem must be handled to maintain the given accuracy throughout the entire day where the environment is expected to change.

Which solution is appropriate is difficult to say. If time was not a factor, it would be interesting to implement all analysed approaches and compare them. However, especially the solutions of using the signaltransmitters, hence the infrastructure, to help change the radio map seem relevant for BlueCAML. The reason for this is that these would make use of the potential of having complete access to the dedicated infrastructure, which can be a problem for systems using an existing infrastructure where system administrators might not want this allowed.

Verification Using UPPAAL

In general, UPPAAL turned out as a great tool with a relatively intuitive design. Also, the opportunity of a command line utility gave rise to execute the model checker remotely on the application server at our disposal. This was very helpful since the model checking process for certain properties need to run for several hours to determine whether or not it was satisfied. However, some deficiencies were observed. Some of these even seem to be based on design and architectural flaws of the tool which, if corrected, would increase the potential of UPPAAL even more. A deficiency, was the discovery made that UPPAAL apparently only allow a single thread for the state simulations. This significantly reduces the amount of resources which can put into model checking as this only allow a single CPU for executing this thread.

The GUI of UPPAAL offered a very useful visualisation of the protocol and how the various components of the system communicate and take actions accordingly. In this regard, UPPAAL has proven useful for finding deficiencies in the communication design. The visualisation could, however, also be done using an appropriate UML diagram, but it does not offer the possibility of actually simulating the system which proved important.

In terms of adding the time constraints to the model in UPPAAL, this turned out to not add a large degree of significance to the project. The purpose was to see whether the mobile applications could request radio maps and upload fingerprints within a given time frame provided that concurrent mobile applications are using the same signaltransmitter as well as back-end. However, due to the time constraints being simple in the model, it was possible with some effort to manually confirm whether actions for a small number of mobile devices and signaltransmitters could be conducted within a given time frame.

During the simulation of the communication protocol, it was discovered that the initial model was too complex and suffered from state-space explosion which resulted in UPPAAL crashing due to insufficient resources. The lesson learned from this is that emphasis should be

kept in modelling as simple as possible to avoid state-space explosion leading to practically unsolvable problems.

Security and Privacy Concerns

Security has not been a concern in developing BlueCAML, therefore multiple security mechanisms should undoubtedly be implemented before deployment.

Currently, the back-end lacks authentication and authorisation mechanisms meaning that everyone with access to it can create, edit and delete the resources, such a fingerprints. As a solution to this, mechanisms as the ones used in Easy Clocking (Sw701b, 2009), such as a login system, can be implemented. This would ensure that the signaltransmitters would have to provide correct credentials before being able to change the resources.

In addition, data sent on the communication paths is not secured, meaning that the data potentially could be a victim of packet interception or injection. Whether this is an actual problem depends on the users' opinion, meaning that a survey should be conducted in order to clarify this. The reason for this is that the data sent does not contain any confidential data, other than RSSI values and the users current position or a radio map. In the authors' opinion the only data that to some extent could be confidential would be the user's current position, but as mentioned if it should be confidential should be the users decision. To secure the communication paths, encryption on all data sent could be enforced.

Extendibility

BlueCAML is developed with the special purpose of presenting the user with her position and to let the user help maintaining the radio map. However, one could imagine parts of BlueCAML being integrated in other systems. For instance, a system like StreamSpin contains modules for indoor and outdoor positioning with Wi-Fi and GPS, respectively. If the accuracy achievable by BlueCAML using Bluetooth would be of interest for StreamSpin, e.g. to provide indoor room level positioning, it would be relatively easy to extend StreamSpin with the functionality of BlueCAML. First of all, the StreamSpin client is written in .NET CF, and secondly BlueCAML uses the same coordinate system and map format as StreamSpin.

Additionally, BlueCAML could be extended itself by means of providing context-aware information, navigation etc. to the users. Also, it could be extended with the possibility of using the signaltransmitters to track Bluetooth devices similar to what BLIP Systems (BLIP, 2010) does.

Finally, the idea of letting the users help maintain the radio map could be adopted by other indoor positioning systems.

Part VI

Appendices

Appendix Contents

A

Quality Factor Definitions

Product Operations	
Correctness	Extent to which a program satisfies its specifications and fulfils the user's mission objectives.
Reliability	Extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform a function.
Usability	Effort required to learn, operate, prepare input, and interpret output of a program.
Integrity	Extent to which access to software or data by unauthorised persons can be controlled.
Availability	The extent to which the program is operable in respect to the total running time.
Durability	The extent to which data is persistent when first stored in the program.
Product Revision	
Flexibility	Effort required to modify an operational program.
Maintainability	Effort required to locate and fix a fault in an operational program.
Testability	Effort required to test a program to ensure it performs its intended function.
Scalability	Extent to which the program can handle information when the program is scaled according to some factor.
Product Transition	
Portability	Effort required to transfer a program from one hardware configuration and/or environment to another.
Reusability	Extent to which a program can be used in other applications related to the packaging and scope of the functions that the program performs.
Interoperability	Effort required to couple one system with another.

Table A.1: Quality factor definitions (van Vliet, 2008, c. 6). Availability, durability and scalability are defined according to our understanding of them.

B

Usability Test Plan

The introduction read for the test subjects, in the usability test, is described below:

Usability Test Introduction Welcome to this test session. During the tests it is important that you are aware that we are testing the system, not you. To improve the system you must think aloud and honestly let us know what you think.

BlueCAML is a system whose purpose is to make indoor positioning possible using a Bluetooth enabled mobile device. It works by measuring signal strengths, to a number of signaltransmitters, thereby composing a fingerprint for a specific location. Afterwards, this fingerprint is compared to a database of fingerprints made from an initialisation phase. This database is called a radio map. Basically, the position related to the best matching fingerprint is returned.

Over time, the radio map gets outdated and hence must be updated to maintain the accuracy of the position estimates. In BlueCAML, the users are asked to help updating the database.

After the introduction is read, to the test users they are asked to do the following tasks one by one:

- Start the application.
- What is the initial thoughts of the GUI?
- The radio map needs to be maintained. Navigate to this functionality.
- What is represented by the different presented symbols?
- The location at the left must be maintained.
- Now the radio map must be updated.
- The above tasks must be repeated with no connection to the back-end.
- What are the general consideration of the maintenance phase?

C

Benchmarker Tool

To show the difference in accuracy using different techniques with different parameters, a benchmarking tool is developed. In its current state the tool makes available the KNN position estimation technique where K can be varied. Also, it makes available the option of combining KNN with Weighted Graphs.

Giving the tool the positions of the actual walked path and the collected fingerprints, it can simulate the position estimates using the Location Engine from the BlueCAML mobile application. The tool can both depict the walked path and the matched positions on a map and give a cumulated frequency of the accuracy using the given settings.

Figure C.1 shows a screenshot of the tool.

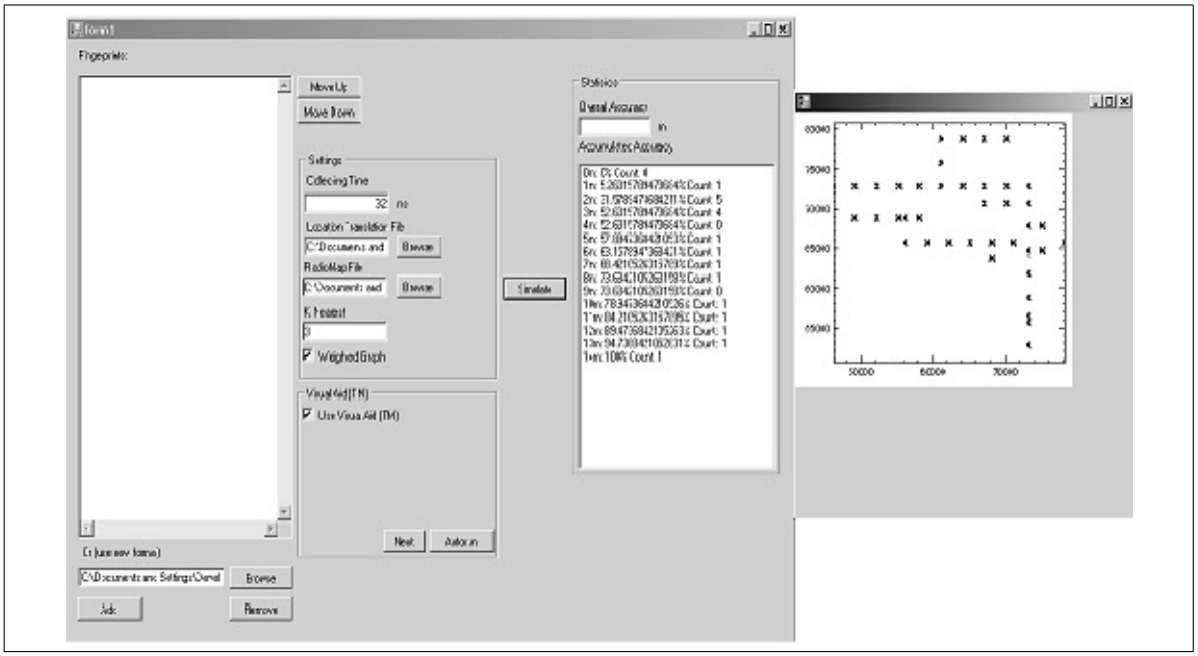


Figure C.1: Screenshot of the Benchmarker tool.

D

Tailored Scrum

The following appendix depicts the sprint backlog and the burndown chart for sprint 1, both taken from Bananascrum which is an online Scrum tool.

D.1 Sprint Backlog

Figure D.1 depicts the sprint backlog from the first sprint of BlueCAML.

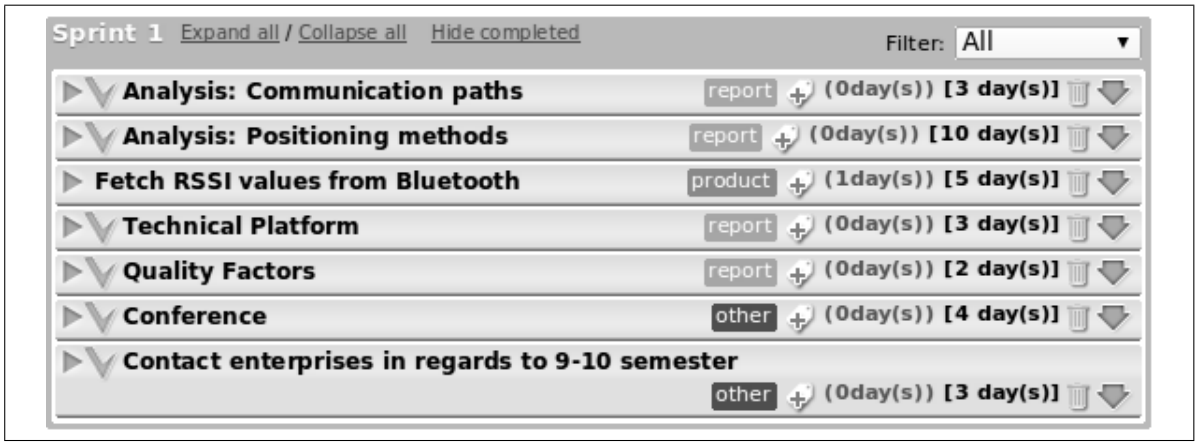


Figure D.1: The sprint backlog of the first sprint in the development of BlueCAML.

As shown the backlog both contains product and report relevant items.

D.2 Burndown Chart

Figure D.2 depicts the burn down chart from the first sprint.

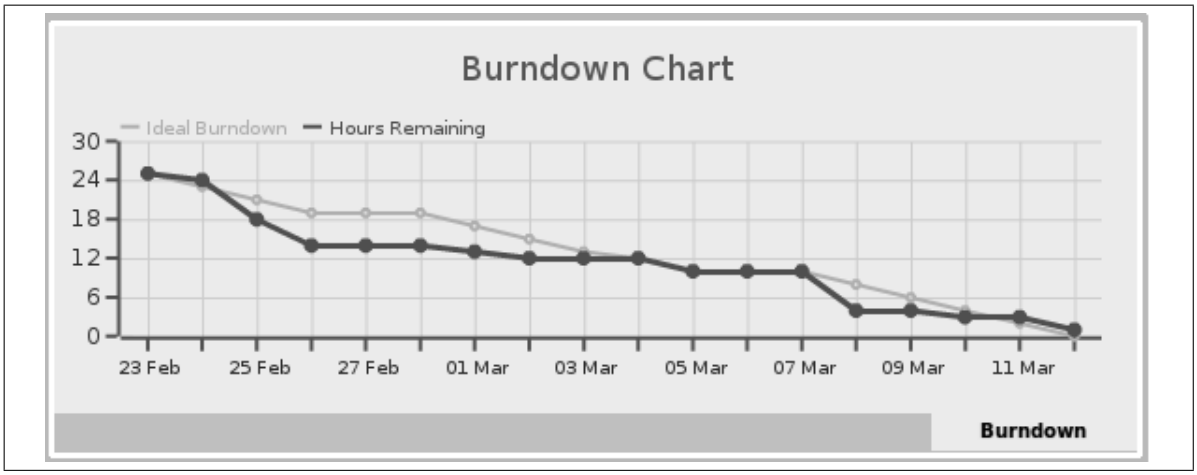


Figure D.2: The burn down chart from the first sprint of BlueCAML.

Bibliography

Apple(2009). Objective-c. http://developer.apple.com/iphone/library/documentation/General/Conceptual/DevPedia-CocoaCore/ObjectiveC.html#//apple_ref/doc/uid/TP40008195-CH43 (15th February 2).

Atira(2009). Scrum and agile methods the real world. https://intranet.cs.aau.dk/fileadmin/user_upload/Education/Courses/2009/SOE/Slides/lecture14_atira.pdf (15. October 2009).

Bahl, P. and Padmanabhan, V.N.(2000). Radar: an in-building rf-based user location and tracking system. *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2, 775–784 vol.2. doi: 10.1109/INFCOM.2000.832252. URL http://dx.doi.org/10.1109/INFCOM.2000.832252.

Barnes, D.(2010). Fundamentals of microsoft .net compact framework development for the microsoft .net framework developer. http://msdn.microsoft.com/en-us/library/aa446549.aspx (16th February 2010).

Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., and Yi, W.(1996). UPPAAL—a tool suite for automatic verification of real-time systems. *Hybrid Systems III*, 232–243.

BLIP(2010). Marketing and tracking solutions. http://www.blipsystems.com/ (17th February 2010).

Bose, A. and Foh, C.H.(2007). A practical path loss model for indoor wifi positioning enhancement.

Burnette, E.(2007). Jobs: No java for you. http://blogs.zdnet.com/Burnette/?p=238 (15th February 2010).

Calzolai, F., De Nicola, R., Loreti, M., and Tiezzi, F.(2008). TAPAs: a tool for the analysis of process algebras. *Transactions on Petri Nets and Other Models of Concurrency I*, 54–70.

Canalys(2010). Canalys quarterly research highlights. http://www.canalys.com/pr/2010/r2010021.html (15th February 2010).

Cerrada, R.(2008). Model view presenter part i – building it from scratch. http://www.cerquit.com/blogs/post/MVP-Part-I-e28093-Building-it-from-Scratch.aspx (5. May 2010).

CityGML(2010). Hompage of citygml: Current news. http://www.citygml.org/1524.

Cohn, M.(2004). *User Stories Applied: For Agile Software Development.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Corporation, M.(2008). Model-view-presenter pattern. http://msdn.microsoft.com/en-us/library/cc304760.aspx (5. May 2010).

David, A. and Amnell, T.(2002). Uppaal2k: Small tutorial.

DD-WRT(2009). Wl-500g premium v2. http://www.dd-wrt.com/wiki/index.php/WL500G_Premium_v2 (4. May 2010).

DetDigitaleByggeri(2010). The Digital Construction: The Ten Construction-Requirements (danish site). http://www.detdigitalebyggeri.dk/public_client/de-ti-bygherrekrav (accessed 10 March 2010).

DEVELOPMENT, I.(2008). alchemo-for-iphone. http://innaworks.com/alcheMo-for-iPhone.html (18. Februrary 2010).

Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., and Burgelman, J.C.(2001). Scenarios for ambient intelligence in 2010.

Evans, D.(1996). Static detection of dynamic memory errors. *SIGPLAN Not.*, 31(5), 44–53. doi:http://doi.acm.org/10.1145/249069.231389.

Fenton, N.(2010). Bayes rule. http://www.eecs.qmul.ac.uk/~norman/BBNs/Bayes_rule.htm (1st March 2010).

Ferial Shayeganfar, A.A. and Tjoa, A.M.(2008). A smart indoor navigation solution based on building information model and google android.

Fischer, G., Dietrich, B., and Winkler, F.(2004). Bluetooth indoor localization system. *Proceedings of the 1st Workshop on Positioning, Navigation and Communication (WPNC'04).*

Fitzek, F.H.P.(2009). Mobile phone programming - and its application to wireless networking aug 2007.

Foundation, S.(2010). Symbian c++ in a nutshell. http://developer.symbian.org/wiki/index.php/Symbian_C%2B%2B_in_a_Nutshell (15th February 2).

Fowler, M.(2006a). Passive View. http://martinfowler.com/eaaDev/PassiveScreen.html (accessed 3 May 2010).

Fowler, M.(2006b). UI Architectures. http://martinfowler.com/eaaDev/uiArchs.html (accessed 5 May 2010).

Frost, C., Jensen, C.S., Larsen, H.W., Luckow, K.S., Madsen, L.S., and Weisberg, A.(2008). Privacy module for streamsp!n. Technical report, Aalborg University, Department of Computer Science.

Hansen, R.(2010). Conversations with rene hansen, ph.d. candidate, department of computer science, aalborg university.

Hansen, R. and Thomsen, B.(2009). Efficient and accurate wlan positioning with weighted graphs.

Heckmann, R. and Ferdinand, C.(2005). Verifying safety-critical timing and memory-usage properties of embedded software by abstract interpretation. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, 618–619. IEEE Computer Society Washington, DC, USA.

Hinton, A., Kwiatkowska, M., Norman, G., and Parker, D.(2006). PRISM: A tool for automatic verification of probabilistic systems. *Tools and Algorithms for the Construction and Analysis of Systems*, 441–444.

Honkavirta, V., Perala, T., Ali-loytty, S., and Piche, R.(2009). A comparative survey of wlan location fingerprinting methods. *Wpnc: 2009 6Th Workshop On Positioning, Navigation And Communication, Proceedings*, 243–251.

HTC(2010). Htc - products - htc touch diamond - specification. `http://www.htc.com/www/product/touchdiamond/specification.html` (25. May 2010).

Jensen, C.S., Bested, M., Hansen, J.L., Larsen, H.W., and Dinh, L.(2007). Chess dominator. Technical report, Aalborg University.

Johnson, J.I.(2009). Windows mobile power management. `http://www.codeproject.com/KB/windows/WiMoPower1.aspx` (25. May 2010).

King, T., Kopf, S., Haenselmann, T., Lubberger, C., and Effelsberg, W.(2006). Compass: A probabilistic indoor positioning system based on 802.11 and digital compasses. URL `http://www.informatik.uni-mannheim.de/pi4/publications/King2006g.pdf`.

Krug, S.(2005). *Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition*. New Riders Press.

Larman, C.(2003). *Agile and Iterative Development*. A Managers Guide.

Li, B., Wang, Y., Lee, H., Dempster, A., and Rizos, C.(2005). Method for yielding a database of location fingerprints in wlan. *Communications, IEE Proceedings-*, 152(5), 580 – 586. doi:10.1049/ip-com:20050078.

Li, B., Salter, J., Dempster, A.G., and Rizos, C.(2007). Indoor positioning techniques based on wireless lan.

Lionel M. Ni, Yunhao Liu, Y.C.L. and Patil, A.(2004). Landmarc: Indoor location sensing using active rfid. *Online Document*. `www.cs.ust.hk/~liu/Landmarc.pdf`.

Liu, H., Darabi, H., Banerjee, P., and Liu, J.(2007). Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6), 1067 –1080. doi:10.1109/TSMCC.2007.905750.

Malekpour, A., Ling, T.C., and Lim, W.C.(2008). Location determination using radio frequency rssi and deterministic algorithm. *Communication Networks and Services Research, Annual Conference on*, 0, 488–495. doi:http://doi.ieeecomputersociety.org/10.1109/CNSR.2008.32.

Malik, A.(2009). *RTLS For Dummies*. Wiley Publishing, Inc.

Mathur, A.P.(2008). *Foundations of Software Testing*. Addison-Wesley Professional.

Microsoft(2010). Code Samples for Windows Mobile. `http://msdn.microsoft.com/en-us/library/bb158662.aspx` (accessed 15 May 2010).

Munk-Madsen, A., Mathiassen, L., Nielsen, P.A., and Stage, J.(2000). *Object Oriented Analysis and Design*. Marko.

Mysaifu(2010a). Mysaifu jvm - a free java virtual machine for windows mobile. `http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html` (16th February 2010).

Mysaifu(2010b). Mysaifu jvm - a free java virtual machine for windows mobile. `http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html` (16th February 2010).

NCover(2010). NCover. `http://www.ncover.com/` (accessed 21 May 2010).

OpenWrt(2010). Openwrt. `http://openwrt.org/` (4. May 2010).

OracleSDN(2010). Java me technology. `http://java.sun.com/javame/technology/index.jsp` (15th February 2010).

RailsGuides(2009). Getting started with Ruby on Rails. `http://guides.rubyonrails.org/getting_started.html` (accessed 5 May 2010).

Rodriguez, A.(2008). Restful web services: The basics.

SIG, B.(2007). Specification of the Bluetooth System Version 2.1+ EDR, Volume 0.

Sommerville, I.(1999). *Software Engineering*. Addison Wesley.

Stephan Mäs, W.R. and Wang, F.(2008). Conception of a 3d geodata web service for the support of indoor navigation with gnss.

Sun(2005). Programming the blackberry with j2me. `http://developers.sun.com/mobility/midp/articles/blackberrydev/` (15th February 2010).

Sw701b(2009). Easy clocking - a system for automatically clocking in and out employees. Technical report, Aalborg University, Department of Computer Science. Christian Frost and Casper Svenning Jensen and Kasper Søe Luckow.

Symbian(2010). Java me quick start - symbian developer community. `http://developer.symbian.org/wiki/index.php/Java_ME_Quick_Start` (15th February 2010).

University, U. and University, A.(2006). Uppaal. `http://www.uppaal.com/` (4. May 2010).

Valgrind(2010). Valgrind Home. `http://valgrind.org` (accessed 16 May 2010).

van Vliet, H.(2008). *Software Engineering: Principles and Practice*. John Wiley & Sons.

Weisberg, A., Frost, C., Larsen, H., Luckow, K., Myrup, M., and Poulsen, T.(2007). Student assessment system. Technical report, Aalborg University, Department of Computer Science.

Wikipedia(2010). Wikipedia:protection policy. `http://en.wikipedia.org/wiki/Protection_policy` (11. March 2010).

Woodings, R., Joos, D., Clifton, T., and Knutson, C.(2002). Rapid heterogeneous connection establishment: Accelerating Bluetooth inquiry using IrDA. In *Proc. of WCNC*. Citeseer.

Yao, P. and Durant, D.(2010). *Programming .NET compact framework 3.5; 2nd ed.* Pearson Education, Boston, MA. Order from outside CERN via Inter Library Loan.

Yeh, L.W., Hsu, M.S., Lee, Y.F., and Tseng, Y.C.(2009). Indoor localization: Automatically constructing today's radio map by irobot and rfids.

Yin, J., Yang, Q., and Ni, L.(2005). Adaptive Temporal Radio Maps for Indoor Location Estimation. *In Proceedings of the 3rd Annual IEEE International Conference on Pervasive Computing and Communications (IEEE PerCom 2005)*, 85–94.

Šikšnys, L.(2010). Work by laurynas Šikšnys, ph.d. candidate, department of computer science, aalborg university.